第8章 控 件 窗 口

控件窗口是允许用户与应用程序进行交互的按钮、列表框和滚动条。当创建对话框时，使用对话框编辑器定义的控件窗口也随之创建。然而，有时某个控件的外观和位置并不尽如人意，这时便可以对控件进行处理，本章的实例包括：

实例28 自己绘制的控件，将看到在使用与控件相关的其他额外开销时如何绘制自己的控件。

实例29 在窗口标题中添加按钮，将为一个窗口的标题添加一个按扭。

实例30 添加热键控件，将为用户的应用程序添加热键。

8.1 实例28：自己绘制的控件

1. 目标

自己绘制一个控件窗口。在这个实例中，将绘制一个列表框，选择的列表项目将产生缩进，如图8-1所示。

图8-1 自己绘制的列表框

2. 策略

首先使用属主绘制风格 (owner-drawn style)创建用户自己的控件，然后从 MFC 控件类中派生自己的控件类，接下来重载 DrawItem()成员函数，在此可以用 MFC的 CDC 类和它的成员函数来绘制控件，本实例中将绘制一个列表控件窗口。

3. 步骤

1) 创建一个新的列表框类

用ClassWizard从CListBox中派生出一个新的类。

2) 绘制列表框控件

用 ClassWizard 重载CListBox类的DrawItem()函数。

首先在DrawItem()函数中封装(wrap)从CDC类对象的DRAWITEMSTRUCT结构中所获得的设备环境的句柄：

```
void CWzdListBox::DrawItem( LPDRAWITEMSTRUCT lpDIS )
{
   // get our device context and rectangle to draw to
   CDC dc;
   dc.Attach( lpDIS -> hDC );
   CRect rect( lpDIS -> rcItem );
```

下一步如何处理取决于怎样绘制这个控件。这可以有很多种情况。对于这个实例，只是简单地采用CDC::DrawItem()来绘制列表框上的项目。但如果某个列表项被鼠标选取，那么将用系统的高亮颜色来绘制该项目，同时它还将缩进 10个像素。

根据是否被选择来确定并设置列表项的前景色和背景颜色：

```
// if our item is selected, then set colors accordingly
//     and fill in background
COLORREF bk = dc.GetBkColor();
COLORREF fg = dc.GetTextColor();
if ( lpDIS -> itemState & ODS_SELECTED )
{
   bk = ::GetSysColor( COLOR_HIGHLIGHT );
   fg = ::GetSysColor( COLOR_HIGHLIGHTTEXT );
}
dc.SetTextColor( fg );
```

这一步为这个列表项绘制背景颜色：

```
CBrush brush( bk );
dc.FillRect( &rect, &brush );
```

接下来用CDC::DrawText()函数来绘制这个列表项的文本，如果选择这行，那么缩进文本的开始10个像素：

```
if ( lpDIS -> itemState & ODS_SELECTED )
   rect.left += 10;
int nBkMode = dc.SetBkMode( TRANSPARENT );

CString str;
GetText( lpDIS -> itemID,str );
dc.DrawText( str, &rect, DT_LEFT|DT_VCENTER );
```

注意是从列表框控件本身获取要绘制的文本，一般来说，列表框并不能为一个属主绘制控件维持该列表。但可以将这个控件设置为 Has String属性，它将在内部继续维护该列表，这将允许继续使用这个控件类中的 AddString()函数。

最后需要在DrawItem()函数中进行清除工作：

```
dc.SetTextColor( fg );
dc.SetBkMode( nBkMode );
dc.Detach();
```

此列表框控件类的完整程序代码可参考本实例结尾的程序清单 —— 列表框类。

3) 使用新的列表框类

为使用这个新的控件类，首先用对话框编辑器为对话框模板添加列表框，确保设置Owner-Drawn Fixed和Has Strings属性。

如果还没有这样做，则使用 ClassWizard为该模板创建一个对话框。

使用ClassWizard为对话框类添加控件成员变量，然后使用用户的新类名替代 Class Wizard自动添加的ClistBox类名，同时将该定义设置在 {{}}括号之外，使得Class Wizard不会被它弄糊涂，这个列表框将在下次调用 UpdateData(FALSE)时被用户的列表框类子类化。

4. 注意

■ 除Windows列表框控件之外，其他可以属主绘制的控件包括：组合框、扩展的组合框、列表控件、按钮、标签控件。列表控件和组合框控件都允许用户设置其中每一个项目的尺寸，这可以通过Dialog Editor设置Owner-Drawn Variable风格来实现。

■ 当使用Owner-Drawn Variable风格时，必须重载用户新控件类的 MeasureItem()成员函数并返回每一个项目所希望的尺寸。重载 MeasureItem()成员函数的例子可参考实例 7。

■ 如果使用Dialog Editor还将该控件设置为Sort风格，则必须同时在新的控件类中重载SortItem()成员函数。

■ 实际上，在这里“属主绘制”这个词有一点用词不当，从技术上来说，本实例中的控件是自己绘制的控件，“属主绘制”最初的含义是某个控件的属主 (owner)，例如一个对话框，可以绘制它所拥有的那个控件。然而，在这个实例中，控件发出的用于通知它的属主绘制(WM_DRAWITEM)时间已到的消息，将会反射回到该控件的 MFC类并在此进行处理，因此从技术上说，这个正被MFC类绘制的控件拥有这个控件。然而，由于在 MFC世界中这两种情况被认为是同一个术语，本例中的控件应该被认为是它自己绘制自己或自绘制 (self-drawn)的。关于消息反射的详细内容，请参阅第1章。

■ 当只是编写一个WM_PAINT消息处理函数来绘制控件时，为什么要费心考虑控件的属主绘制风格呢？原因是想使绘制工作和控件的其他功能同步。例如，一个属主绘制组合框将会告诉用户何时绘制它的列表框，而且更重要的是，它将会告诉用户绘制列表框的位置，使得鼠标在屏幕上单击该位置时，将会与该控件上相应的项目相关联。甚至一个属主绘制的按钮控件将通知用户何时绘制一个已按下的或未按下的按钮，使得与当单击它时所创建的命令消息相符。

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，然后再单击Wzd菜单命令，打开一个带有列表框的对话框。注意到在单击列表框中的任何一个列表项时都将会导致文本框中的相应一行缩进。

6. 程序清单 —— 列表框类

```
#if !defined( AFX_WZDLISTBOX_H__41500055_E450_11D1_9B7D_00AA003D8695__INCLUDED_ )
#define AFX_WZDLISTBOX_H__41500055_E450_11D1_9B7D_00AA003D8695__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CWzdListBox window

class CWzdListBox : public CListBox
{
// Construction
public:
    CWzdListBox();

// Attributes
public:

// Operations
public:

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

```
    // {{AFX_VIRTUAL( CWzdListBox )
public:
    virtual void DrawItem( LPDRAWITEMSTRUCT lpDrawItemStruct );
    // }}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdListBox();

// Generated message map functions
protected:
    // {{AFX_MSG( CWzdListBox )
    // }}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

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

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

#endif
    // !defined( AFX_WZDLISTBOX_H__41500055_E450_11D1_9B7D_00AA003D8695__INCLUDED_ )
// WzdListBox.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdListBox.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdListBox

CWzdListBox::CWzdListBox()
{
}

CWzdListBox::~CWzdListBox()
{
}


BEGIN_MESSAGE_MAP( CWzdListBox, CListBox )
    // {{AFX_MSG_MAP( CWzdListBox )
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
//////////////////////////////////////////////////////////////////////////
// CWzdListBox message handlers

void CWzdListBox::DrawItem( LPDRAWITEMSTRUCT lpDIS )
{
   // get our device context and rectangle to draw to
   CDC dc;
   dc.Attach( lpDIS -> hDC );
   CRect rect( lpDIS -> rcItem );

   // if our item is selected, then set colors accordingly
   //    and fill in background
   COLORREF bk = dc.GetBkColor();
   COLORREF fg = dc.GetTextColor();
   if ( lpDIS -> itemState & ODS_SELECTED )
   {
      bk = ::GetSysColor( COLOR_HIGHLIGHT );
      fg = ::GetSysColor( COLOR_HIGHLIGHTTEXT );
   }
   dc.SetTextColor( fg );
   CBrush brush( bk );
   dc.FillRect( &rect, &brush );

   // draw text
   if ( lpDIS -> itemState & ODS_SELECTED )
      rect.left += 10;
   int nBkMode = dc.SetBkMode( TRANSPARENT );

   CString str;
   GetText( lpDIS -> itemID,str) ;
   dc.DrawText(str, &rect, DT_LEFT|DT_VCENTER);

   // cleanup
   dc.SetTextColor( fg );
   dc.SetBkMode( nBkMode );
   dc.Detach();
}
```

8.2 实例29：在窗口标题中添加按钮

1. 目标

在一个窗口的标题栏中添加一个按钮，如图 8-2所示。

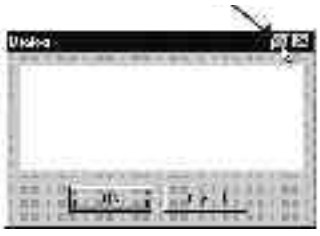

图8-2 标题栏中的一个按钮

2. 策略

从技术上来说，本例将要创建的按钮不是一个控件窗口。实际上要创建的是一个“假的”按钮，该按钮将由用户自己绘制并且处理鼠标消息。系统通过响应两个消息 (WM_NCPAINT和WM_ACTIVATE)来绘制窗口标题，在每个消息使用CDC类的绘制功能之后，用户将对本例中的按钮进行绘制，同时还将处理另一个消息 (WM_NCLBUTTONDOWN)

来确定用户是否按下本例中的“ 按钮 ”。

3. 步骤

1) 为标题按钮设置位图

用位图编辑器(Bitmap Editor)创建两个位图：一个用于按钮弹起时，另一个用于按钮按下时。可以在位图边界使用高亮度和阴影颜色来达到这一效果，背景颜色采用灰色 —— 装载位图时代替这种颜色。对于一个用于通常窗口标题中常规尺寸的按钮，可以设置其尺寸为 16×16像素，对于一个工具栏窗口的小尺寸的按钮，可设置其尺寸为 12×12像素。

在自己的对话框类中的构造函数中，将这两个位图装载到嵌入的 CBitmap成员变量中。利用随书附带光盘上为该实例提供的 CWzdBitmap类，可以将当前用于按钮的系统颜色替代为灰色背景色，还可以将按钮的状态初始化为未按下 (unpressed)。

```
CWzdDialog::CWzdDialog( CWnd* pParent /* = NULL*/ )
   : CDialog( CWzdDialog::IDD, pParent )
{
   // {{AFX_DATA_INIT( CWzdDialog )
   // }}AFX_DATA_INIT
   m_bPressed = FALSE;
   m_bitmapPressed.LoadBitmapEx ( IDB_PRESSED_BITMAP, TRUE );
   m_bitmapUnpressed.LoadBitmapEx ( IDB_UNPRESSED_BITMAP, TRUE );
}
```

注意 这个位图类(CWzdBitmap)在本系列丛书中的第一本书《Visual C++ MFC编程实例》中创建。

2) 绘制标题按钮

用ClassWizard为该类添加WM_NCPAINT和WM_ACTIVATE消息处理函数，从这两个消息处理函数中调用一个DrawButton()函数，该函数将用于绘制按钮。接下来将编写 DrawButton () 函数的代码：

```
void CWzdDialog::OnNcPaint()
{
   // draw caption first
   CDialog::OnNcPaint();

   // then draw button on top
   DrawButton();
}

void CWzdDialog::OnActivate( UINT nState, CWnd* pWndOther,
   BOOL bMinimized )
{
   CDialog::OnActivate( nState, pWndOther, bMinimized );

   DrawButton();
}
```

创建 DrawButton() 函数，这个函数将决定绘制哪个按钮位图并且使用如下所示的StretchBlt()函数绘制它：

```
void CWzdDialog::DrawButton()
```

```
{
    // if window isn't visible or is minimized, skip
    if ( !IsWindowVisible() || IsIconic() )
        return;

    // get appropriate bitmap
    CDC memDC;
    CDC* pDC = GetWindowDC();
    memDC.CreateCompatibleDC( pDC );
    memDC.SelectObject( m_bPressed ? &m_bitmapPressed :
        &m_bitmapUnpressed );

    // get button rect and convert into non-client area coordinates
    CRect rect, rectWnd;
    GetButtonRect(rect);
    GetWindowRect(rectWnd);
    rect.OffsetRect( -rectWnd.left, -rectWnd.top );

    // draw it
    pDC -> StretchBlt( rect.left, rect.top, rect.Width(),
        rect.Height(), &memDC, 0, 0, m_bitmapPressed.m_Width,
        m_bitmapPressed.m_Height, SRCCOPY );

    memDC.DeleteDC();
    ReleaseDC( pDC );
}
```

注意到从GetWindowDC()中获得了设备环境，这样设备环境就可以绘制到整个窗口，包括标题所在的非客户区。这里使用 StretchBlt()是因为用户并不能确定使用何种分辨率来绘制对话框。同时，本例使用了另一个称为 GetButtonRect()局部辅助函数来确定按钮的尺寸，该函数将在下一步创建。

创建GetButtonRect()函数以确定按钮的尺寸，这将依靠系统告知自己的按钮的相应尺寸，并使用GetSystemMetrics()函数来获得这些尺寸：

```
void CWzdDialog::GetButtonRect( CRect& rect )
{
    GetWindowRect(&rect);

    // for small caption use SM_CYDLGFRAME
    rect.top += GetSystemMetrics(SM_CYFRAME)+1;

    // for small caption use SM_CYSMSIZE
    rect.bottom = rect.top + GetSystemMetrics( SM_CYSIZE )-4;

    // for small caption use SM_CXDLGFRAME
    rect.left = rect.right - GetSystemMetrics( SM_CXFRAME ) -
        ( 2 * // set to number of buttons already in caption + 1
        GetSystemMetrics( SM_CXSIZE ) )-1;
        // for small caption use SM_CXSMSIZE

    // for small caption use SM_CXSMSIZE
    rect.right = rect.left + GetSystemMetrics(SM_CXSIZE)-3;
}
```

3) 允许用户单击按钮

为了确定用户是否单击本例中的“按钮”，使用ClassWizard为这个类添加WM_NCLBUTTONDOWN消息处理函数，它可以用来检查用户是否已经单击了标题栏，如果是，则进一步检查是否单击其中的按钮所在之处。注意在这里又用到了 GetButtonRect()函数：

```
void CWzdDialog::OnNcLButtonDown( UINT nHitTest, CPoint point )
{
   if ( nHitTest == HTCAPTION )
   {
      // see if in area we reserved for button
      CRect rect;
      GetButtonRect( rect );
      if ( rect.PtInRect( point ) )
      {
         m_bPressed = !m_bPressed;
         DrawButton();
      }
   }
   CDialog::OnNcLButtonDown( nHitTest, point );
}
```

此对话框类的完整程序代码可参考本实例结尾的程序清单 —— 对话框类。

4. 注意

■ 本例中不仅需要处理WM_NCPAINT 消息来绘制按钮，而且还需要处理WM_ACTIVATE消息。这是因为当激活窗口时，WM_ACTIVATE消息将导致系统重绘标题颜色(当窗口激活时，窗口标题的颜色从暗灰色变为亮色)。

■ 确保在系统绘制标题然后绘制用户自己的“按钮”，本例中所完成的一切就是在系统所绘制的内容之上再绘制一个位图。

■ 本实例演示了一个保持被按下状态的按钮，为了在单击该按钮时进行切换，应该在处理WM_NCLBUTTONDOWN时一直绘制表示按钮“按下”的位图，然后添加一个新的WM_NCLBUTTONUP消息处理函数来绘制表示按钮“弹起”的位图。

■ 同样的方法可以用于任何一种控件，只要不介意为每一种控件添加各自的逻辑代码。

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，然后再单击Wzd菜单命令，打开一个在标题中有按钮的对话框。单击该按钮，使之保持按下或未按下状态。

6. 程序清单 —— 对话框类

```
#if !defined( AFX_WZDDIALOG_H__A76FC804_D3BD_11D1_9B67_00AA003D8695__INCLUDED_ )
#define AFX_WZDDIALOG_H__A76FC804_D3BD_11D1_9B67_00AA003D8695__INCLUDED_

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

#include "WzdBitmap.h"
```

```
//////////////////////////////////////////////////////////////////////////
// CWzdDialog dialog

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

// Dialog Data
    // {{AFX_DATA( CWzdDialog )
    enum { IDD = IDD_WZD_DIALOG };
    // NOTE: the ClassWizard will add data members here
    // }}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 )
    afx_msg void OnNcPaint();
    afx_msg void OnNcLButtonDown( UINT nHitTest, CPoint point );
    afx_msg void OnActivate( UINT nState, CWnd* pWndOther, BOOL bMinimized );
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL m_bPressed;
    CWzdBitmap m_bitmapPressed;
    CWzdBitmap m_bitmapUnpressed;

    void DrawButton();
    void GetButtonRect( CRect &rect );
};

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

#endif
    // !defined( AFX_WZDDIALOG_H__A76FC804_D3BD_11D1_9B67_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

#define HTBUTTON 32
/////////////////////////////////////////////////////////////////////////////
// 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_bPressed = FALSE;
   m_bitmapPressed.LoadBitmapEx( IDB_PRESSED_BITMAP,TRUE );
   m_bitmapUnpressed.LoadBitmapEx( IDB_UNPRESSED_BITMAP,TRUE );
}


void CWzdDialog::DoDataExchange( CDataExchange* pDX )
{
   CDialog::DoDataExchange( pDX );
   // {{AFX_DATA_MAP( CWzdDialog )
   // NOTE: the ClassWizard will add DDX and DDV calls here
   // }}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP( CWzdDialog, CDialog )
   // {{AFX_MSG_MAP( CWzdDialog )
   ON_WM_NCPAINT()
   ON_WM_NCLBUTTONDOWN()
   ON_WM_ACTIVATE()
   // }}AFX_MSG_MAP
END_MESSAGE_MAP()

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


void CWzdDialog::OnNcPaint()
{
   // draw caption first
   CDialog::OnNcPaint();

   // then draw button on top
   DrawButton();
}

// caption is also redrawn with WM_ACTIVATE message for color change
void CWzdDialog::OnActivate( UINT nState, CWnd* pWndOther, BOOL bMinimized )
```

```
{
    CDialog::OnActivate( nState, pWndOther, bMinimized );

    DrawButton();
}

void CWzdDialog::OnNcLButtonDown( UINT nHitTest, CPoint point )
{
    if ( nHitTest == HTCAPTION )
    {
        // see if in area we reserved for button
        CRect rect;
        GetButtonRect( rect );
        if ( rect.PtInRect( point ) )
        {
            m_bPressed = !m_bPressed;
            DrawButton();
        }
    }
    CDialog::OnNcLButtonDown( nHitTest, point );
}

void CWzdDialog::GetButtonRect( CRect& rect )
{
    GetWindowRect( &rect );

    // for small caption use SM_CYDLGFRAME
    rect.top += GetSystemMetrics( SM_CYFRAME )+1;

    // for small caption use SM_CYSMSIZE
    rect.bottom = rect.top + GetSystemMetrics( SM_CYSIZE )-4;

    // for small caption use SM_CXDLGFRAME
    rect.left = rect.right - GetSystemMetrics( SM_CXFRAME ) -
        // set to number of buttons already in caption + 1
        ( 2 *
        // for small caption use SM_CXSMSIZE
        GetSystemMetrics( SM_CXSIZE ) )-1;

    // for small caption use SM_CXSMSIZE
    rect.right = rect.left + GetSystemMetrics( SM_CXSIZE )-3;
}

void CWzdDialog::DrawButton()
{
    // if window isn't visible or is minimized, skip
    if ( !IsWindowVisible() || IsIconic() )
        return;

    // get appropriate bitmap
    CDC memDC;
    CDC* pDC = GetWindowDC();
    memDC.CreateCompatibleDC( pDC );
```

```
    memDC.SelectObject( m_bPressed ? &m_bitmapPressed : &m_bitmapUnpressed );

    // get button rect and convert into non-client area coordinates
    CRect rect, rectWnd;
    GetButtonRect( rect );
    GetWindowRect( rectWnd );
    rect.OffsetRect( -rectWnd.left, -rectWnd.top );

    // draw it
    pDC -> StretchBlt( rect.left, rect.top, rect.Width(), rect.Height(), &memDC,
        0, 0, m_bitmapPressed.m_Width, m_bitmapPressed.m_Height, SRCCOPY );

    memDC.DeleteDC();
    ReleaseDC( pDC );
}
```

8.3 实例30：添加热键控件

1. 目标

在自己的应用程序中添加热键。

2. 策略

热键是一种小有名气的功能，它允许用户与当前处于非激活状态的应用程序进行交互。当用户按下一个热键时，经常是Shift、Control、Alt和其他一些字母键的组合，则用户自己的应用程序可以被激发或可以接收一个WM_HOTKEY消息来处理。

Dialog Editor允许在对话框模板上添加一个热键控件，但这并不是一个真正的热键控件。相反，它实际上更像是一个热键编辑框，将各种键 (例如Shift/Alt/Control)的组合转换成它们的等效文本(当这个控件具有当前焦点时，按下 Shift键和F键将会使Shift+F在控件上出现)。而这个控件的输出将可以直接用于Windows API中并注册一个热键。

必须在系统中注册热键，这样当按下热键时才会激活自己的应用程序，这里将使用WM_SETHOTKEY完成注册。当按下一个热键组合时，将会把 WM_SETHOTKEY消息发送给用户的应用程序，这里将使用 ::RegisterHotKey()函数。

3. 步骤

1) 为用户提示热键

用Dialog Editor在用户自己的对话框模板中添加一个热键编辑框。

用Class Wizard为这个对话框模板类添加热键控件成员变量。

用下面的代码来从Hotkey控件中获得热键的键码：

```
WORD m_wVkCode;
WORD m_wModifier;

m_ctrlHotKey.GetHotKey( m_wVkCode, m_wModifier );
```

2) 注册用于激活应用程序的热键

发送一个带有以上所返回的热键值的 WM_RETHOTKEY消息到主窗口：

```
AfxGetMainWnd() -> SendMessage(WM_SETHOTKEY,
    ( WPARAM )MAKEWORD( m_wVkCode,m_wModifier ) );
```

3) 注册发送WM_HOTKEY消息的热键

为通过RegisterHotKey()函数来使用热键编辑框控件的输出，要求首先将由 GetHotKey()函数返回的组合键(modifier)转换成RegisterHotKey()函数可以使用的值：

```
// must translate hot key control's returned modifier to
//     RegisterHotKey format
UINT mod = 0;
if ( m_wModifier&HOTKEYF_ALT ) mod| = MOD_ALT;
if ( m_wModifier&HOTKEYF_CONTROL ) mod| = MOD_CONTROL;
if ( m_wModifier&HOTKEYF_SHIFT ) mod| = MOD_SHIFT;
if ( m_wModifier&HOTKEYF_EXT ) mod| = MOD_WIN;
m_wModifier = mod;
```

注册热键：

```
::RegisterHotKey(
   AfxGetMainWnd() -> m_hWnd,          // window to receive hot-key
                                       //     notification
   1234,                               // identifier of hot key
   m_wModifier,                        // key-modifier flags
   m_wVkCode                           // virtual-key code
);
```

通过在相应的非子窗口 (Mainframe和Child Frame是唯一合适的窗口)中手工添加代码处理WM_HOTKEY消息：

```
// to the message map
ON_MESSAGE( WM_HOTKEY,OnHotKey )

LRESULT CMainFrame::OnHotKey( WPARAM wParam, LPARAM lParam )
{
   wParam;        // id specified in RegisterHotKey()
   // (in this example: 1234)
   lParam;        // virtual key code and modifiers of the hotkey
   return 0L;
}
```

4. 注意

■ 用户在允许输入哪些键作为热键上必须小心，因为组合键将取代任何其他应用程序的快捷键的优先权。例如，如果用户指定 ALT+S组合键作为一个热键，即使该热键组合可能是当前使用的应用程序用于保存工作的快捷键，它也不会发挥其快捷键作用，而只是以热键方式或者激发用户自己的应用程序或者发出一个 WM_HOTKEY消息。

■ 热键和快捷键的区别在于热键将允许用户和自己的应用程序之间进行交互，甚至于在应用程序处于非激活时也可以进行交互。

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，然后再单击Wzd菜单命令，打开一个允许输入热键组合的对话框。单击 Make Active Application和OK，然后激活另一个应用程序并且再此输入热键组合，则这个应用程序将会被再次激活。单击 Send WM_HOTKEY Message并在CMainFrame的OnHotKey()函数中设置一个断点，然后按下热键组合可看到应用程序在这一断点终止。