第 5 章　菜单、控件条和状态栏

菜单和控件条(如工具栏)代表了一种用户与应用程序之间交互的基本方式。 AppWizard会自动地给应用程序添加通用菜单、工具栏和状态栏。但是，这些菜单、工具栏和状态栏与用户在Developer Studio中使用的菜单和控件条相比就显得非常“暗淡无光”。然而通过一些小小的努力，就可以给应用程序添加外观与通用菜单相同的菜单和控件条。本章包括以下具体实例：

实例7　在菜单中添加图标，可以模仿在 Developer Studio的菜单中看到的图标。

实例8　调整命令条外观，可以模仿在 Developer Studio中看到的工具栏的外观。

实例9　创建可编程工具栏，可以模仿在 Developer Studio中工具栏的“感觉”—用户可以从工具栏按钮库中给工具栏选择并设置一个表示其功能的名字。

实例10　在对话框应用程序中添加一个工具栏和状态栏，这里将给对话框应用程序手工添加一个工具栏和状态栏。

实例11　给弹出菜单添加一个位图标记，并沿着弹出菜单的边界绘制一幅位图。

实例12　在工具栏中添加下拉按钮 (Dropdown Button)，创建两个工具栏按钮，用它们来创建看起来像是从工具栏下拉 (drop down)的弹出菜单。

实例13　在状态栏中添加一个图标，本实例在状态栏中添加一个状态图标。

实例14　使用伸缩条 (Rebar)，给应用程序添加一个伸缩条，并用工具栏和对话框条填充它。

5.1　实例7：在菜单中添加图标

1. 目标

模仿Developer Studio中的菜单，在每个菜单项旁边显示一个图标，而且这些菜单都有一个工具栏按钮，如图5-1所示。

在本例中，将在与工具条功能等效的菜单旁边绘制相同的工具条图标

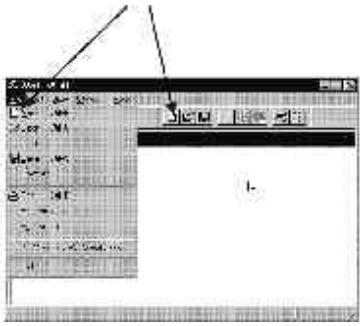

图5-1　在菜单中添加图标

2. 策略

上面所看到的菜单是一个自绘制 (owner-drawn)的菜单。为了创建一个自绘制的菜单，必须在一开始时就在每个菜单项中设置一个选项，告诉系统这是一个自绘制的菜单。由于在应用程序中没有提供菜单编辑器，因此必须动态地完成这些事情。然后自绘制菜单就向自己的窗口发送两个消息(WM_MEASUREITEM和WM_DRAWITEM)并通过绘制菜单项时来处理该消息。本例将所有这些功能都封装到从MFC的CMenu类派生的菜单类中。这个新类中的InitMenu()函数不仅为每个菜单项进行标记表示该菜单为自绘制菜单，而且还将菜单命令和它们的相对应工具栏匹配起来，同时还将在绘制菜单项时使用匹配的工具栏按钮位图。

3. 步骤

1) 创建一个来自CMenu的新类

单击Developer Studio中的Insert和New Class菜单命令来打开New Class对话框。从Class type组合框中选择 Generic Class。输入一个新类名，并由 CMenu派生该类。此处不用 Class Wizard完成这些工作的原因是Class Wizard通常不支持CMenu。正因为如此，必须手工向该类加入其他东西。

本实例仍然允许使用菜单编辑器来编辑应用程序的菜单。因此这个新菜单类的 InitMenu()函数可以完成以下一些工作。首先，它可以将应用程序中所有的菜单项转换为自绘制的菜单项。其次，由于自绘制的菜单不能保存自己的文本名，因此必须将每个菜单项的名字保存到自己的数组中，还要在数组中保存每个菜单项使用像素绘制时所需的大概尺寸，以便以后快速绘制。同时，在应用程序的工具栏中将每个菜单项命令的 ID号和其相对应的命令ID号匹配起来。指向工具栏的指针将作为InitMenu()函数的一个参数来传送。下面将开始具体工作。

2) 向新的菜单类添加自己的InitMenu( )

给该类添加一个InitMenu()函数，同时给它传送一个指向应用程序的 CWnd类的指针和一个用于封装(wraps)其菜单的CMenu指针。

```
void CWzdMenu::InitMenu( CWnd *pWnd, CMenu *pMenu, UINT idb,
   CToolBar *pToolBar )
{
```

如下所示，遍历所有在Cmenu指针中发现的菜单项：

```
CDC *pDC = pWnd -> GetDC();

// for all submenus
CMenu *pSubMenu = NULL;
for ( int i = 0; i < ( int )pMenu -> GetMenuItemCount(); i++ )
{
   pSubMenu = pMenu -> GetSubMenu( i );
   if ( pSubMenu )
   {
      for ( int j = 0;j < ( int )pSubMenu -> GetMenuItemCount(); j++ )
      {
         // if not a separator...
         UINT id = pSubMenu -> GetMenuItemID( j );
         if ( id )
         {
            // if already ownerdrawn, escape
            if ( pSubMenu ->
               GetMenuState( j,MF_BYPOSITION )&MF_OWNERDRAW )
            {
               pWnd -> ReleaseDC( pDC );
               return;
            }
```

注意这里忽略了分隔菜单项(separator menu item)，而且如果某个菜单项已经是自绘制的，则由于该菜单已经进行了转换将被忽略。

不幸的是，一旦菜单项成为自绘制的，它将不会保留名字字符串 (name string)。与其保留

一个存放菜单项名字的单个数组，还不如将这些名字保存在由菜单编辑器生成的菜单中。出于这个目的，需要创建一个新的名为 MENUITEM的结构。在本步骤，创建一个 MENUITEM 的新实例，并将菜单项的名字字符串复制到其中：

```
// fill in MENUITEM
MENUENTRY *pEntry = new MENUENTRY;
pEntry -> id = id;
pSubMenu -> GetMenuString( j,pEntry -> str,MF_BYPOSITION );
```

自绘制的菜单不再处理快捷键项目 (菜单中可以通过键盘访问的带有下划线的字母)。因此必须自己动手添加快捷键功能，为此必须保存在创建这个菜单之初利用菜单编辑器所选择的字母。可以寻找 “ & ” 之后的字母，然后将该字符作为快捷键方式保存在 MENUITEM中：

```
int k = pEntry -> str.Find( '&' );
pEntry -> chr = 0;
if ( k >= 0 )
   pEntry -> chr = pEntry -> str[k+1];
pEntry -> chr& = ~0x20;    // make upper case
```

以像素为单位计算绘制菜单项所需的尺寸，同时将计算结果保存在 MENUITEM中。以后在处理WM_MEASUREITEM消息时，要用到这些值:

```
pEntry -> size = pDC -> GetTextExtent( pEntry -> str,
   pEntry -> str.GetLength() );
pEntry -> size.cx += BUTTON_WIDTH + 2;
pEntry-> size.cy = BUTTON_HEIGHT + 6;
```

这一步将该菜单命令和工具栏中与其相对应的命令匹配起来。为此需要查看工具栏中该命令的ID号。如果找到，将它保存起来以便在需要绘制这个菜单项时使用:

```
pEntry -> inx = -1;
for ( int m = 0;m < pToolBar -> GetToolBarCtrl().GetButtonCount();m++ )
{
   int inx;
   UINT idx,x;
   pToolBar -> GetButtonInfo( m,idx,x,inx );
   if ( id = idx )
   {
      pEntry -> inx = inx;
      break;
   }
```

继续下一步，使该菜单项成为一个自绘制的菜单项：

```
// modify menu item to be owner drawn
pSubMenu -> ModifyMenu( id, MF_BYCOMMAND | MF_ENABLED | MF_OWNERDRAW,
   id, ( LPCTSTR )pEntry );
```

同时还要装载工具栏的位图以便以后绘制图标时用。此时将使用在本系列书的前面书中创建的CwzdBitmap类使图像的背景可以转换为菜单的现有颜色，这样就可以使得它们看起来是透明的：

```
m_bitmap.LoadBitmapEx(idb, TRUE);
```

系统为应用程序发送两个消息以允许绘制一个自绘制菜单。第一个消息是 WM_

MEASUREITEM，该消息用来告诉系统菜单中的每个菜单项的尺寸大小，系统由此可以确定整个菜单的大小。第二个消息 WM_DRAWITEM是为单个菜单项发送使得允许使用标准的Windows绘图工具来绘制菜单。

3) 为新菜单类增加MeasureItem()函数

Classwizard还是不支持CMenu，因此仍然需要为该类手工添加WM_MEASUREITEM消息处理函数。

该处理函数将返回以像素为单位的被请求菜单项的大小，系统需要该值以确定整个菜单的宽度。由于前面已经用InitMenu()函数计算出了这些像素值，因而可以使用这些数值如下:

```
// size of our menu item
void CWzdMenu::MeasureItem( LPMEASUREITEMSTRUCT lpMIS )
{
   MENUENTRY *pEntry = ( MENUENTRY * )lpMIS -> itemData;
   lpMIS -> itemWidth = pEntry -> size.cx;
   lpMIS -> itemHeight = pEntry -> size.cy;
}
```

4) 为该新菜单类增加DrawItem()函数

为新的菜单类手工添加WM_DRAWITEM消息处理函数:

```
void CWzdMenu::DrawItem( LPDRAWITEMSTRUCT lpDIS )
{
```

封装从MFC类的DRAWITEMSTRUCT结构所得到的信息以启动该消息处理函数，如下所示：

```
CDC dc;
dc.Attach( lpDIS -> hDC );              // device context to draw to
CRect rect( lpDIS -> rcItem );          // rectangular size of menu item
```

接下来绘制菜单项的背景。这里得到的设备环境已经绘制了标准的背景颜色。但是，如果鼠标在该项上时，需要转换背景色和文本色：

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

// fill in background
CBrush brush( bk );
dc.FillRect( &rect, &brush );
```

如果将要绘制的菜单项设置为禁用，那么需要将该菜单项的文本和相关联的图标变灰。这就要用到CDC::DrawState()函数，该函数与一个处理实际绘图工作的回调函数一起工作，因此这里DrawItem()所要做的就是设置DrawState()函数以调用该回调函数:

```
// get enabled/disabled state and draw appropriately
UINT nState = DSS_NORMAL;
```

```
if ( lpDIS -> itemState & ODS_DISABLED )
{
    nState = DSS_DISABLED;
}
dc.DrawState( rect.TopLeft(), rect.Size(), DrawStateProc,
   ( LPARAM )lpDIS, nState, ( HBRUSH )NULL );
```

最后清除设备环境以完成DrawItem()函数：

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

5) 为该新菜单类增加CDC::DrawState()回调函数

接下来创建CDC::DrawState()所需要的回调函数并再次为其封装设备环境句柄和矩形来启动该函数：

```
BOOL CALLBACK DrawStateProc( HDC hdc, LPARAM lData, WPARAM wData,
   int cx, int cy )
{
   CDC dc;
   LPDRAWITEMSTRUCT lpDIS = ( LPDRAWITEMSTRUCT )lData;
   dc.Attach( hdc );
   MENUENTRY *pEntry = ( MENUENTRY * )lpDIS -> itemData;
   CRect rect( 0, 0, cx, cy );
```

如果在以上的InitMenu()中发现了一个匹配的工具栏，这一步将在菜单项 CDC::BitBlt函数中绘制与之相关联的位图：

```
if ( pEntry -> inx != -1 )
   {
      CDC memDC;
      memDC.CreateCompatibleDC( &dc );
      memDC.SelectObject( pEntry -> pBitmap);
      dc.BitBlt( rect.left, rect.top, BUTTON_WIDTH, BUTTON_HEIGHT,
         &memDC, pEntry -> inx*BUTTON_WIDTH, 0, SRCCOPY );
      memDC.DeleteDC();
   }
```

下一步查看该菜单项是否被设置为复选，如果是，则绘制一个特定的复选标记，该复选标记是在图标编辑器(Icon Editor)中创建的位图。如果在最后一步没有绘制位图，该复选标记将单独存在：

```
HICON hIcon;
if ( lpDIS -> itemState & ODS_CHECKED &&
   ( hIcon = AfxGetApp() -> LoadIcon( IDI_CHECK_ICON ) ) )
{
   dc.DrawIcon( rect.left, rect.top, hIcon );
}
```

下面用DrawText()>为该菜单项绘制文本字符串，此时使用 DrawText()函数使“ & ”之后的字符产生下划线表示该字符标识的按键作为快捷键。 DrawText()还将对齐文本并扩展标签：

```
   rect.left += BUTTON_WIDTH + 2;
```

```
    dc.DrawText( pEntry -> str, &rect,
        DT_LEFT|DT_EXPANDTABS|DT_VCENTER );

    dc.Detach();
    return TRUE;
}
```

6) 为新菜单类增加MenuChar()函数

正如以上提到的，自绘制菜单不再处理快捷键命令 (菜单项中的下划线字符)。为了手工处理该命令，需要为新菜单类增加 WM_MENUCHAR消息处理函数。此时使用前面在 InitMenu() 中保存的快捷键信息来确定所应当触发的菜单项：

```
LRESULT CWzdMenu::MenuChar( UINT nChar )
{
    nChar& = ~0x20;    // make uppercase
    // try to find char in current menu list
    for ( POSITION pos = m_CurrentMenuList.GetHeadPosition();pos; )
    {
        MENUENTRY *pEntry = ( MENUENTRY * )
            m_CurrentMenuList.GetNext( pos );
        if ( pEntry -> chr = nChar )
        {
            AfxGetMainWnd() -> SendMessage( WM_COMMAND,pEntry -> id );
            return( MAKELONG( 0,1 ) );
        }
    }
    return(0);
}
```

为了查看该菜单类的完整程序可以参考程序清单 —— 菜单类。

为了在应用程序中添加新菜单类，首先将其嵌入主框架类，然后将为该主框架类添加消息处理函数以调用该菜单类的成员函数。

7) 将新菜单类合并到应用程序

CMainFrame类内部嵌入的新菜单类如下所示：

```
// add CWzdMenu to Mainfrm.h
private:
    CWzdMenu m_menu;
```

这一步要将CWzdMenu类连接到主菜单。用 ClassWizard添加5个消息处理函数(可以再次使用 ClassWizard)。所对应的窗口消息是： WM_INITMENU、WM_INITPOPMENU、WM_MEASUREITEM、WM_DRAWITEM和WM_MENUCHAR。

在每个增加到 CMainFrame的消息处理函数内调用对应的新菜单类的成员函数，如下所示：

```
void CMainFrame::OnInitMenu( CMenu* pMenu )
{

    CMDIFrameWnd::OnInitMenu( pMenu );

    m_menu.InitMenu( this,pMenu,IDR_MAINFRAME,&m_wndToolBar );
```

```
}

void CMainFrame::OnInitMenuPopup( CMenu* pPopupMenu, UINT nIndex,
    BOOL bSysMenu )
{
    CMDIFrameWnd::OnInitMenuPopup( pPopupMenu, nIndex, bSysMenu );

    m_menu.InitMenuPopup();

}

void CMainFrame::OnMeasureItem( int nIDCtl,
    LPMEASUREITEMSTRUCT lpMeasureItemStruct )
{
    if ( !nIDCtl ) m_menu.MeasureItem( lpMeasureItemStruct );

    CMDIFrameWnd::OnMeasureItem( nIDCtl, lpMeasureItemStruct );
}

void CMainFrame::OnDrawItem( int nIDCtl,
    LPDRAWITEMSTRUCT lpDrawItemStruct )
{
    if ( !nIDCtl ) m_menu.DrawItem( lpDrawItemStruct );

    CMDIFrameWnd::OnDrawItem( nIDCtl, lpDrawItemStruct );
}

LRESULT CMainFrame::OnMenuChar( UINT nChar, UINT nFlags, CMenu* pMenu )
{
    return m_menu.MenuChar( nChar );
}
```

4. 注意

由于本例只是一个实例，所以省去了一些精华内容。菜单项中的加速键描述应该以右对齐方式绘制。当鼠标停留在该菜单项之上时，图标将使用焦点窗口 (一个细的边框)绘制来进行修改。

禁用(Disabled)菜单项是用CDC::DrawState()以一种浮雕灰色绘制的。位图也显示浮雕效果，但是在某些情况下，它们的整个背景色将被转换，从而使得它们所具有的位图都显得模糊。如果不希望出现这种效果，可以通过使整个位图的背景色设置为白色来进行纠正。为此需要第二次装载工具栏位图并将背景色转换成白色。这就需要修改本实例中 CWzdBitmap类。

5. 使用光盘时注意

执行随书附带光盘上的工程时，可以注意到任何在工具栏中具有相对应命令的菜单项，都具有一个相邻该工具栏的图标。

6. 程序清单 —— 菜单类

```
#if !defined( AFX_WZDMENU_H__18606914_D521_11D1_9B69_00AA003D8695__INCLUDED_ )
#define AFX_WZDMENU_H__18606914_D521_11D1_9B69_00AA003D8695__INCLUDED_

#if _MSC_VER >= 1000
```

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

// WzdMenu.h : header file
//
#include "afxtempl.h"

struct MENUENTRY
{
    CString str;
    UINT  id;
    UINT  chr;
    CSize size;
    int   inx;
};

/////////////////////////////////////////////////////////////////////////////
// CWzdMenu window

class CWzdMenu : public CMenu
{
// Construction
public:
    CWzdMenu();
    virtual ~CWzdMenu();

// Attributes
public:

// Operations
public:
    void InitMenu( CWnd *pWnd, CMenu *pMenu, UINT idb, CToolBar *pToolBar );
    void InitMenuPopup();
    void MeasureItem( LPMEASUREITEMSTRUCT lpMIS );
    void DrawItem( LPDRAWITEMSTRUCT lpDIS );
    LRESULT MenuChar( UINT nChar );

// Implementation
public:

private:
    CBitmap m_bitmap;
    CList<MENUENTRY*,MENUENTRY*> m_FullMenuList;
    CList<MENUENTRY*,MENUENTRY*> m_CurrentMenuList;
};

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

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

```
#endif
    // !defined( AFX_WZDMENU_H__18606914_D521_11D1_9B69_00AA003D8695__INCLUDED_ )
// WzdMenu.cpp : implementation file
//
#include “stdafx.h”
#include “wzd.h”
#include “WzdMenu.h”
#include “WzdProject.h”

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

BOOL CALLBACK DrawStateProc( HDC hdc, LPARAM lData, WPARAM wData, int cx,
    int cy );

////////////////////////////////////////////////////////////////////////////
// CWzdMenu

CWzdMenu::CWzdMenu()
{
}

CWzdMenu::~CWzdMenu()
{
    while ( !m_FullMenuList.IsEmpty() )
    {
        delete m_FullMenuList.RemoveHead();
    }
}

void CWzdMenu::InitMenu( CWnd *pWnd, CMenu *pMenu, UINT idb, CToolBar *pToolBar )
{
    CDC *pDC = pWnd -> GetDC();

    // for all submenus
    CMenu *pSubMenu = NULL;
    for (int i = 0; i < ( int )pMenu -> GetMenuItemCount(); i++ )
    {
        pSubMenu = pMenu -> GetSubMenu( i );
        if ( pSubMenu )
        {
            for ( int j = 0;j < ( int )pSubMenu -> GetMenuItemCount(); j++ )
            {
                // if not a separator...
                UINT id = pSubMenu -> GetMenuItemID( j );
                if ( id )
                {
                    // if already ownerdrawn, escape
```

```
                if ( pSubMenu ->
                    GetMenuState( j,MF_BYPOSITION )&MF_OWNERDRAW )
                {
                    pWnd -> ReleaseDC( pDC );
                    return;
                }

                // fill in MENUITEM
                MENUENTRY *pEntry = new MENUENTRY;
                pEntry -> id = id;
                pSubMenu -> GetMenuString( j,pEntry -> str,MF_BYPOSITION );
                int k = pEntry -> str.Find( '&' );
                pEntry -> chr = 0;
                if ( k >= 0 )
                    pEntry -> chr = pEntry -> str[k+1];
                pEntry -> chr& = ~0x20;    // make upper case
                pEntry -> size = pDC ->
                    GetTextExtent( pEntry -> str,pEntry -> str.GetLength() );
                pEntry -> size.cx += BUTTON_WIDTH + 2;
                pEntry -> size.cy = BUTTON_HEIGHT + 2;
                pEntry -> inx = -1;
                for ( int m = 0;m < pToolBar ->
                    GetToolBarCtrl().GetButtonCount();m++ )
                {
                    int inx;
                    UINT idx,x;
                    pToolBar -> GetButtonInfo( m,idx,x,inx );
                    if ( id = idx )
                    {
                        pEntry -> inx = inx;
                        pEntry -> pBitmap = &m_bitmap;
                        break;
                    }
                }

                // add MENUITEM to full list
                m_FullMenuList.AddTail( pEntry );

                // modify menu item to be owner drawn
                pSubMenu -> ModifyMenu( id, MF_BYCOMMAND | MF_ENABLED |
                    MF_OWNERDRAW, id, ( LPCTSTR )pEntry );
            }
        }
    }
  }
  m_bitmap.LoadBitmapEx( idb,TRUE );
  pWnd -> ReleaseDC( pDC );
}

void CWzdMenu::InitMenuPopup()
{
```

```
    // empty current menu list
    m_CurrentMenuList.RemoveAll();
}

// size of our menu item
void CWzdMenu::MeasureItem( LPMEASUREITEMSTRUCT lpMIS )
{
    MENUENTRY *pEntry = ( MENUENTRY * )lpMIS -> itemData;
    lpMIS -> itemWidth = pEntry -> size.cx;
    lpMIS -> itemHeight = pEntry -> size.cy;
}

// draw our menu item
void CWzdMenu::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
    COLORREF bk = dc.GetBkColor();
    COLORREF fg = dc.GetTextColor();
    if ( lpDIS -> itemState & ODS_SELECTED )
    {
        bk = ::GetSysColor( COLOR_HIGHLIGHT );
        fg = ::GetSysColor( COLOR_HIGHLIGHTTEXT );
    }
    dc.SetTextColor( fg );

    // fill in background
    CBrush brush( bk );
    dc.FillRect( &rect, &brush );

    // draw text withhout a background
    int nBkMode = dc.SetBkMode( TRANSPARENT );

    // get enabled/disabled state and draw appropriately
    UINT nState = DSS_NORMAL;
    if ( lpDIS -> itemState & ODS_DISABLED )
    {
        nState = DSS_DISABLED;
    }
    dc.DrawState( rect.TopLeft(),rect.Size(),DrawStateProc,
        ( LPARAM )lpDIS,nState,( HBRUSH )NULL );

    // add to current menu list
    MENUENTRY *pEntry = ( MENUENTRY * )lpDIS -> itemData;
    m_CurrentMenuList.AddTail( pEntry );

    // cleanup
```

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

BOOL CALLBACK DrawStateProc( HDC hdc, LPARAM lData, WPARAM wData, int cx, int cy )
{
    CDC dc;
    LPDRAWITEMSTRUCT lpDIS = ( LPDRAWITEMSTRUCT )lData;
    dc.Attach( hdc );
    MENUENTRY *pEntry = ( MENUENTRY * )lpDIS -> itemData;
    CRect rect( 0,0,cx,cy );

    // draw bitmap, if any
    if ( pEntry -> inx != -1 )
    {
        CDC memDC;
        memDC.CreateCompatibleDC( &dc );
        memDC.SelectObject( pEntry -> pBitmap );
        dc.BitBlt( rect.left, rect.top, BUTTON_WIDTH,BUTTON_HEIGHT, &memDC,
            pEntry -> inx*BUTTON_WIDTH, 0, SRCCOPY );
        memDC.DeleteDC();
    }

    // check it, if required
    HICON hIcon;
    if ( lpDIS -> itemState & ODS_CHECKED &&
        ( hIcon = AfxGetApp() -> LoadIcon( IDI_CHECK_ICON ) ) )
    {
        dc.DrawIcon( rect.left, rect.top, hIcon );
    }
    // draw text
    rect.left += BUTTON_WIDTH + 2;
    dc.DrawText( pEntry -> str, &rect, DT_LEFT|DT_EXPANDTABS|DT_VCENTER );

    dc.Detach();
    return TRUE;
}

// draw our menu item
LRESULT CWzdMenu::MenuChar( UINT nChar )
{
    nChar& = ~0x20;    // make uppercase
    // try to find char in current menu list
    for ( POSITION pos = m_CurrentMenuList.GetHeadPosition();pos; )
    {
        MENUENTRY *pEntry = ( MENUENTRY * )m_CurrentMenuList.GetNext( pos );
        if ( pEntry -> chr = nChar )
        {
            AfxGetMainWnd() -> SendMessage( WM_COMMAND,pEntry -> id );
            return( MAKELONG( 0,1 ) );
```

```
        }
    }
    return( 0 );
}
```

5.2 实例8：调整命令条外观

1. 目标

使工具栏具有与Developer Studio中的工具栏同样的外观，如图5-2所示。

2. 策略

使用TBSTYLE_FLAT工具栏风格使工具栏具备扁平按钮外观。在处理发送到工具栏的WM_PAINT消息时，自己绘制移动条（grabber）。这里将用ClassWizard把所有的功能封装进工具栏类的内部。

图5-2 命令条外观

注意 最新的VC++6.0版本已经增加了一个工具栏风格，该风格自动地在工具栏上绘制提手条(gripper bar)。该风格为CBRS_GRIPPER，但是它仅仅绘制单提手条而不是如本例和Developer Studio所示的那样绘制双提手条。

3. 步骤

(1) 创建新的工具栏类

用ClassWizard 创建一个从 CToolBarCtrl派生的新工具栏类，然后用文本编辑器用CToolBar取代CToolBarCtrl，因为ClassWizard当前还不能从CToolBar派生新类。

用ClassWizard为新的工具栏类增加WM_PAINT 消息处理函数。在工具栏处于水平时在工具栏的前面绘制提手条，或者，在工具栏垂直停靠于应用程序的主窗口时在其顶部绘制提手条。在工具栏浮动的时候则不绘制提手条。首先允许工具栏控件按通常方式绘制其按钮和边界：

```
void CWzdToolBar::OnPaint()
{
    CToolBar::OnPaint();
```

接下来，如果工具栏浮动则返回：

```
// if floating, no grabber bar
if ( IsFloating() )
    return;
```

创建一个窗口设备环境而不是客户设备环境，这是因为没有非客户区包围工具栏：

```
// to draw to whole window!
CWindowDC dc( this );
```

提手条将使用系统定义的颜色绘制。如果用户在控制面板内更改了系统的颜色方案，那么此时使用控件条编号就不会起作用。有两种系统颜色可用于 3D提手条：高亮度颜色和阴影颜色，用两种颜色创建两种画笔：

```
CPen penHL( PS_SOLID, 2, ::GetSysColor( COLOR_3DHIGHLIGHT ) );
CPen penSH( PS_SOLID, 3, ::GetSysColor( COLOR_3DSHADOW ) );
```

现在根据工具栏的对齐方向，在工具栏的开始处或顶部绘制两条直线：

```
CPen *pPen = dc.SelectObject( &penSH );
if ( GetBarStyle()&CBRS_ORIENT_HORZ )
{
   // draw shadow lines
   rect.OffsetRect( 4,4 );
   dc.MoveTo( rect.left,rect.top );
   dc.LineTo( rect.left,rect.bottom );
   dc.MoveTo( rect.left + 4,rect.top );
   dc.LineTo( rect.left + 4,rect.bottom );

   // draw highlight lines
   dc.SelectObject( &penHL );
   dc.MoveTo( rect.left,rect.top );
   dc.LineTo( rect.left,rect.bottom - 1 );
   dc.MoveTo( rect.left + 4,rect.top );
   dc.LineTo( rect.left + 4,rect.bottom - 1 );
}
else
{
   // draw shadow lines
   rect.OffsetRect( 4,2 );
   dc.MoveTo( rect.left,rect.top );
   dc.LineTo( rect.right,rect.top );
   dc.MoveTo( rect.left,rect.top + 4 );
   dc.LineTo( rect.right,rect.top + 4 );

   // draw highlight lines
   dc.SelectObject( &penHL );
   dc.MoveTo( rect.left,rect.top );
   dc.LineTo( rect.right - 1,rect.top );
   dc.MoveTo( rect.left,rect.top + 4 );
   dc.LineTo( rect.right - 1,rect.top + 4 );
}
dc.SelectObject(pPen);
```

查看新的工具栏类完整代码列表，请参考本实例结尾的程序清单 —— 工具栏类。

2) 实现新工具栏类

在CMainFrame 中用新的工具栏类替代原来的工具栏类。可以在如下的 MainFrm.h文件中完成这项工作。在本实例中将新工具栏命名为：

```
CWzdToolBar  m_wndToolBar;
```

在MainFrm.cpp文件中为工具栏应用 TBSTYLE_FLAT风格，在创建该工具栏之后，按如下方式使用CToolBar::ModifyStyle()函数：

```
if ( !m_wndToolBar.Create( this ) ||
   !m_wndToolBar.LoadToolBar( IDR_MAINFRAME ) )
{
```

```
    TRACE0( "Failed to create toolbar\n" );
    return -1;    // fail to create
}
m_wndToolBar.ModifyStyle( 0, TBSTYLE_FLAT );
```

4. 注意

■ 对工具栏使用平坦按钮风格的时候，很难确定工具栏在哪里结束以及按钮从哪里开始。在拖动工具栏的时候就更困难了。这也就是采用提手条的原因。

■ 伸缩条控件像命令条但实际上根本就不是工具栏。伸缩条是一种控件窗口，它可以包含一个或多个其他的控件窗口，包括工具栏等等。它允许用户在它的控制范围之内移动这些控件窗口。

5. 使用光盘时注意

执行随书附带光盘上的工程时，可以注意到工具栏看起来像 Developer Studio中的工具栏，包括扁平按钮和提手条。

6. 程序清单 —— 工具栏类

```
#if !defined( AFX_WZDTOOLBAR_H__0939E911_E0EC_11D1_9B7A_00AA003D8695__INCLUDED_ )
#define AFX_WZDTOOLBAR_H__0939E911_E0EC_11D1_9B7A_00AA003D8695__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar window

class CWzdToolBar : public CToolBar
{
// Construction
public:
    CWzdToolBar();

// Attributes
public:

// Operations
public:

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

// Implementation
public:
    virtual ~CWzdToolBar();
// Generated message map functions
```

```
protected:
    // {{AFX_MSG( CWzdToolBar )
    afx_msg void OnPaint();
    // }}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

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

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

#endif
    // !defined( AFX_WZDTOOLBAR_H__0939E911_E0EC_11D1_9B7A_00AA003D8695__INCLUDED_ )
// WzdToolBar.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdToolBar.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar

CWzdToolBar::CWzdToolBar()
{
}

CWzdToolBar::~CWzdToolBar()
{
}


BEGIN_MESSAGE_MAP( CWzdToolBar, CToolBar )
    // {{AFX_MSG_MAP( CWzdToolBar )
    ON_WM_PAINT()
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar message handlers

void CWzdToolBar::OnPaint()
```

```
{
    CToolBar::OnPaint();
    // if floating, no grabber bar

    if ( IsFloating() )
        return;
    // draw to whole window!
    CWindowDC dc( this );

    // draw horizontal or vertical grabber bar
    CRect rect;
    GetClientRect( &rect );

    CPen penHL( PS_SOLID,2,::GetSysColor( COLOR_3DHIGHLIGHT ) );
    CPen penSH( PS_SOLID,3,::GetSysColor( COLOR_3DSHADOW ) );


    CPen *pPen = dc.SelectObject( &penSH );
    if ( GetBarStyle()&CBRS_ORIENT_HORZ )
    {
        // draw shadow lines
        rect.OffsetRect( 4,4 );
        dc.MoveTo( rect.left,rect.top );
        dc.LineTo( rect.left,rect.bottom );
        dc.MoveTo( rect.left + 4,rect.top );
        dc.LineTo( rect.left + 4,rect.bottom );

        // draw highlight lines
        dc.SelectObject( &penHL );
        dc.MoveTo( rect.left,rect.top );
        dc.LineTo( rect.left,rect.bottom - 1 );
        dc.MoveTo( rect.left + 4,rect.top );
        dc.LineTo( rect.left + 4,rect.bottom - 1 );
    }
    else
    {
        // draw shadow lines
        rect.OffsetRect( 4,2 );
        dc.MoveTo( rect.left,rect.top );
        dc.LineTo( rect.right,rect.top );
        dc.MoveTo( rect.left,rect.top + 4 );
        dc.LineTo( rect.right,rect.top + 4 );

        // draw highlight lines
        dc.SelectObject( &penHL );
        dc.MoveTo( rect.left,rect.top );
        dc.LineTo( rect.right - 1,rect.top );
        dc.MoveTo( rect.left,rect.top + 4 );
        dc.LineTo( rect.right - 1,rect.top + 4 );
    }

    dc.SelectObject( pPen );
}
```

5.3 实例9：可编程工具栏

1. 目标

允许用户从可选工具栏按钮列表中创建自己的工具栏，如图 5-3所示。

2. 策略

首先要为应用程序的首选项创建新的属性表单以配置工具栏。当某属性页被选中的时候，它将捕获鼠标光标，使得单击屏幕上任何一处都将向该页发送鼠标单击消息。这就允许确定用户是否在该属性页以外单击了工具栏按钮，知道这一点后就可以用CToolBar的普通功能来修改工具栏。属性页中的工具栏按钮实际上只是一个位图图像列表，它们代表工具栏按钮。此外还要为主框架类增加装载和保存修改过的工具栏到系统注册表的功能，以便应用程序下一次运行时可以重新装载到应用程序中。该实例可能是本书中最复杂的实例。

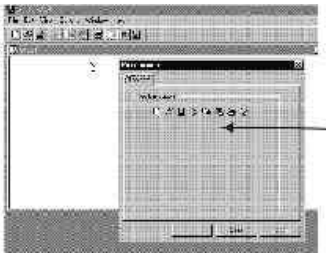

图5-3 根据按钮选择创建工具栏

3. 步骤

1) 创建工具栏属性页

用Dialog Editor创建一个对话框模板并赋予Control风格，为其加入一个简单分组框。

用ClassWizard从这个派生于CPropertyPage的对话框创建一个工具栏属性页。

如下所示，在该属性页中嵌入一个工具栏类和位图类：

```
CToolBar m_ToolBar;
CBitmap m_Bitmap;
```

用ClassWizard为该类增加WM_INITDIALOG消息处理函数。这里将装载应用程序的工具栏和工具栏位图。在本实例中将只使用缺省工具栏。如果应用程序中还有其他更多的工具栏，则只需将其装载即可:

```
BOOL CToolBarPage::OnInitDialog()
{
   CPropertyPage::OnInitDialog();

   // create and load breeder toolbar
   m_ToolBar.Create( this );
   m_ToolBar.LoadToolBar( IDR_MAINFRAME );

   // load breeder toolbar bitmap

   m_bitmap.LoadBitmap( IDR_MAINFRAME );

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

这里装载的工具栏实际上对用户并不可见，只是使用它们作为该属性页可以创建什么按钮的参考。

用ClassWizard为该属性页增加WM_PAINT消息处理函数。这里将绘制可用按钮的图像以创建分组框内的其他工具栏，同时还将使用一些额外处理过程来创建这些按钮图像之间的空间：

```
void CToolBarPage::OnPaint()
{
   CPaintDC dc( this );    // device context for painting

   // display breeder toolbar bitmap
   CDC memDC;
   memDC.CreateCompatibleDC( &dc );
   memDC.SelectObject( &m_bitmap );

   int x = m_xBitmapStart;
   int y = m_yBitmapStart;
   for ( int i = 0; i < m_nButtonCount; i++ )
   {
      dc.BitBlt( x,y,BUTTON_WIDTH,BUTTON_HEIGHT, &memDC,
         i*BUTTON_WIDTH, 0, SRCCOPY );
      x += BUTTON_WIDTH + BUTTON_XSPACING;
   }
   memDC.DeleteDC();
}
```

2) 为该属性页增加WM_SETACTIVE消息处理函数

当属性页打开时，需要获取对鼠标光标的控制权。如果用户在该页以外单击了工具栏按钮，则需要截获该消息以便移动或者删除该按钮。如果没有获取对光标的控制权，则单击工具栏按钮对属性页没有任何效果。使用 ClassWizard为该属性页类增加WM_SETACTIVE消息处理函数。 WM_SETACTIVE消息表明该页被打开，然后用 ::SetCapture()函数来获取光标控制权：

```
BOOL CToolBarPage::OnSetActive()
{
   SetCapture();
   return CPropertyPage::OnSetActive();
}
```

3) 为该属性页增加WM_KILLACTIVE消息处理函数

该页在关闭后，需要释放对鼠标光标的控制权，可以用 ClassWizard增加WM_KILLACTIVE消息处理函数以释放光标：

```
BOOL CToolBarPage::OnKillActive()
{
   ReleaseCapture();
   return CPropertyPage::OnKillActive();
}
```

到目前为止已经有了一个属性页显示工具栏按钮图像，而且在将其打开后可以随时了解用户在屏幕上的单击鼠标行为。这一步需要处理用于表示用户已经在屏幕上进行了单击操作的消息。它可能表明用户将从现有工具栏拖动按钮或者从属性页拖动图像，或者还可能单击了包含该属性页的属性表单上的另一个标签，或者根本就没有单击任何东西。

4) 为属性页增加鼠标按下消息处理函数

用ClassWizard为该类增加WM_LBUTTONDOWN消息处理函数，用于处理用户在屏幕上单击时的表示鼠标已按下的消息：

```
void CToolBarPage::OnLButtonDown( UINT nFlags, CPoint point )
{
```

用CWnd::WindowFromPoint()函数来确定用户单击了什么窗口：

```
m_bMoving = FALSE;
ClientToScreen( &point );
CWnd *pWnd = WindowFromPoint( point );
```

如果用户单击了属性表单或者另一属性表单的标签，就将该消息转发给相应窗口：

```
// if point is parent or child of this property page
if ( pWnd != this && ( pWnd = GetParent() ||
   GetParent() -> IsChild( pWnd ) ) )

{
   CPoint pt( point );
   pWnd -> ScreenToClient( &pt );
   if ( pt.y >= 0 )
   {
      // if clicked on a cliet area, just send
      pWnd ->
         SendMessage( WM_LBUTTONDOWN,nFlags, MAKELONG( pt.x,pt.y ) );
   }
   *else
   {
      // if clicked on a non-client area, we must
      // perform a hit test before sending the click
      UINT ht = pWnd ->
         SendMessage( WM_NCHITTEST,0, MAKELONG( point.x,point.y ) );
      pWnd ->
         SendMessage( WM_NCLBUTTONDOWN,ht,
         MAKELONG( point.x,point.y ) );
   }
   return;
}
```

如果用户单击了现有工具栏上的按钮，就将进入 move button模式。这时通过查看用户是否单击了与应用程序所维护的某一个工具栏相匹配的窗口，来确定用户是否单击了该工具栏按钮。move button模式只意味着需要确定并保存用户单击的按钮。同时还要修改停留在按钮表面的鼠标光标外形：

```
// see if this is a toolbar button
if ( m_pToolBar = GetToolBar( point ) )
{

   m_nButtonMoved = GetButtonIndex( m_pToolBar, point );
   m_pToolBar ->
      GetToolBarCtrl().GetButton( m_nButtonMoved,&m_tbbutton );
```

```
    m_bMoving = TRUE;
    ::SetCursor( m_hMoveCursor );
}
```

如果用户单击了属性页内的按钮图像之一，就必须确定单击了哪一个按钮，将其保存然后再次进入move button模式；

```
// else if this window
else if ( pWnd == this )
{
    ScreenToClient( &point );
    point.Offset( -m_xBitmapStart,-m_yBitmapStart );
    CRect rect( 0,0,(BUTTON_WIDTH+BUTTON_XSPACING )*
        m_nButtonCount,BUTTON_HEIGHT );
    if ( rect.PtInRect( point ) )
    {
        m_nButtonMoved = 0;
        int i = point.x/( BUTTON_WIDTH+BUTTON_XSPACING );
        for ( int j = 0;j < m_nButtonCount;j++ )
        {
            UINT k;
            int l;
            m_ToolBar.GetButtonInfo( j,k,k,l );
            if ( l == i )
            {
                m_nButtonMoved = j;
                break;
            }
        }
        m_ToolBar.GetToolBarCtrl().GetButton(
            m_nButtonMoved,&m_tbbutton );
        m_bMoving = TRUE;
        ::SetCursor( m_hMoveCursor );
        SetCapture();
    }
```

忽略其他的单击行为。参考本实例结尾的程序清单——属性页类，可以查看该消息处理函数代码的完整列表。

用户在什么地方释放了拖动的按钮，将决定应用程序接下来应采取的行为。如果用户在另一个工具栏上释放了它，则应用程序将把该按钮添加到该工具栏。如果用户在另一属性页上释放该按钮，则应该取消该操作，除非按钮是从已有工具栏上拖动，在这种情况下应用程序应该删除该按钮。如果用户在空白区域释放了按钮，则应用程序应当创建新的浮动或者停靠工具栏。

5) 为该属性页增加鼠标单击释放消息

用ClassWizard为该类增加WM_LBUTTONUP消息处理函数：

```
void CToolBarPage::OnLButtonUp( UINT nFlags, CPoint point )
{
```

如果当前不是move button模式，将忽略该消息：

```
if(m_bMoving)
```

```
{
```

如果从现有工具栏的按钮启动 move button模式，则删除该按钮：

```
// delete button from source toolbar
if ( m_pToolBar )
{
    m_pToolBar-> GetToolBarCtrl().DeleteButton( m_nButtonMoved );
}
```

查看用户是否在属性页上释放该按钮。如果是这样则不管，但如果正在拖动已经被删除的已有按钮或者正在属性页以外拖动图像，那么该移动操作应当取消:

```
// if dropped anywhere but toolbar property page
CRect rect;
ClientToScreen (&point );
GetWindowRect( &rect );
if ( !rect.PtInRect( point ) )
{
```

查看用户是否在已有工具栏上释放了按钮，如果这样，就将按钮添加到用户进行释放操作的那个工具栏：

```
// if dropped on existing toolbar, add button to it
CToolBar *pToolBar;
if ( pToolBar = GetToolBar( point ) )
{
    int i = GetButtonIndex( pToolBar,point );
    pToolBar -> GetToolBarCtrl().InsertButton( i,&m_tbbutton );
    UpdateToolBar( pToolBar );
}
```

GetToolBar()实际上是一个为该类创建的小型辅助函数。可以在程序清单 —— 属性页类中找到它。

如果用户没有在已有工具栏上或者属性页上释放按钮，那么需要创建一个新的浮动或者停靠工具栏并将该按钮插入其中:

```
// else create a new toolbar and add our button to it
else
{
    pToolBar = new CToolBar;
    CList<CToolBar*,CToolBar*> *pList =
        ( ( CMainFrame* )AfxGetMainWnd() ) -> GetToolBarList();
    pList -> AddTail( pToolBar );
    pToolBar -> Create( GetParentFrame(),
        WS_CHILD|WS_VISIBLE|CBRS_TOP|CBRS_TOOLTIPS );
    SIZE sizeButton, sizeImage;
    sizeImage.cx = BUTTON_WIDTH;
    sizeImage.cy = BUTTON_HEIGHT;
    sizeButton.cx = sizeImage.cx + BUTTON_XSPACING;
    sizeButton.cy = sizeImage.cy + BUTTON_YSPACING;
    pToolBar -> SetSizes( sizeButton, sizeImage );
    pToolBar -> EnableDocking( CBRS_ALIGN_ANY | CBRS_FLOAT_MULTI );

    // add all possible tool button bitmaps
```

```
    pToolBar ->
        GetToolBarCtrl().AddBitmap( m_nButtonCount,IDR_MAINFRAME );
    // add new button to this new toolbar
    pToolBar -> GetToolBarCtrl().InsertButton( 0,&m_tbbutton );
```

这一步必须确定所创建的工具栏是否可以被停靠到框架窗口，还是只能作为浮动条。这里用下面列出的辅助函数GetToolBarNear()来查看所有的被停靠控件条来了解工具栏是否成行排列，如果成行，再查看工具栏是垂直对齐还是水平对齐。如果新工具栏不能停靠，则将其浮动：

```
UINT nMode;
BOOL bHorz;
if ( GetToolBarNear( point,nMode,bHorz ) )
{
    // dock toolbar centered on cursor
    CSize size = pToolBar -> CalcFixedLayout( FALSE,bHorz );
    if ( bHorz )
        point.x -= size.cx/2;
    else
        point.y -= size.cy/2;
        CRect rectx( point,size );
        GetParentFrame() -> DockControlBar( pToolBar,nMode,&rectx );
        pToolBar -> Invalidate();
        GetParentFrame() -> RecalcLayout();
}
else
{
    GetParentFrame() ->
        FloatControlBar( pToolBar, point, CBRS_ALIGN_TOP );
}
```

如果按钮来自现有的工具栏，则这一步查看该工具栏是否已经没有了按钮，如果没有按钮剩余，则将其删除：

```
// else update source toolbar if any
else if ( m_pToolBar )
{
    UpdateToolBar( m_pToolBar );
    // if no buttons left on toolbar, delete it
    if ( m_pToolBar && m_pToolBar ->
        GetToolBarCtrl().GetButtonCount() == 0 )
    {
        POSITION pos;
        CList<CToolBar*,CToolBar*> *pList =
            ( ( CMainFrame* )AfxGetMainWnd() ) -> GetToolBarList();
        if ( pos = pList -> Find( m_pToolBar ) )
        {
            pList -> RemoveAt( pos );
            m_pToolBar -> m_bAutoDelete = TRUE;
            if ( !m_pToolBar -> IsFloating() )
            {
                m_pToolBar -> SendMessage( WM_CLOSE );
```

```
        }
        else
        {
            CDockBar* pDockBar = m_pToolBar -> m_pDockBar;
            CMiniDockFrameWnd* pDockFrame =
                ( CMiniDockFrameWnd* )pDockBar -> GetParent();
            // won't close till page is gone so hide till then
            pDockFrame -> ShowWindow( SW_HIDE );
            pDockFrame -> SendMessage( WM_CLOSE );
        }
    }
```

6) 为应用程序的属性表选项添加工具栏属性页

如下所示，添加工具栏属性页到应用程序的首选项 (preference)中：

```
void CMainFrame::OnOptionsPreferences()
{
    CPropertySheet sheet( _T( "Preferences" ),this );

    m_pToolBarPage = new CToolBarPage;

    sheet.AddPage( m_pToolBarPage );

    sheet.DoModal();

    delete m_pToolBarPage;
}
```

要查看该属性页类的完整代码列表，可以参考本实例结尾的程序清单 —— 属性表类。

虽然用户现在可以创建自己的工具栏，但是如果不能将工具栏保存起来供以后使用则该功能毫无意义。为此需要为主框架类增加两个新的成员函数。其中一个成员函数在应用程序将要终止时保存工具栏配置，另一个函数则在应用程序首次创建的时候装载该配置。

7) 创建SaveToolbar()函数

在CMainFrame中创建新函数SaveToolbars()，这里首先保存应用程序中的自定义工具栏数目。如下使用CWinApp的WriteProfileInt()函数来保存该值：

```
void CMainFrame::SaveToolbars()
{
    int nNumToolBars = m_ToolBarList.GetCount();
    AfxGetApp() ->
        WriteProfileInt( TOOLBAR_SUMMARY_KEY,TOOLBAR_NUM_KEY,
        nNumToolBars );
```

如果没有任何工具栏需要保存，则立即返回：

```
if ( nNumToolBars )
{
```

接下来遍历应用程序的每个工具栏并为其创建新的系统注册表键：

```
int i = 0;
int idc = IDC_CUSTOM_TOOLBARS;
for ( POSITION pos = m_ToolBarList.GetHeadPosition();pos;i++ )
{
```

```
CToolBar *pToolBar = m_ToolBarList.GetNext( pos );
// create key for this toolbar
CString key;
key.Format( TOOLBAR_KEY,i );

// give toolbar a unique, sequential ID for
//    SaveBarState/LoadBarState
pToolBar -> SetDlgCtrlID( idc++ );
```

然后遍历工具栏上的每个按钮将其信息存入系统注册表：

```
// write number of buttons in this toolbar
int nButtonCount = pToolBar -> GetToolBarCtrl().GetButtonCount();
AfxGetApp() -> WriteProfileInt( key,NUM_OF_BUTTONS_KEY, nButtonCount );
// write info on each button
TBBUTTON *ptbbutton = new TBBUTTON[nButtonCount];
for ( int j = 0;j < nButtonCount; j++ )
{
   pToolBar -> GetToolBarCtrl().GetButton( j,ptbbutton+j );
}
AfxGetApp() -> WriteProfileBinary( key,BUTTON_INFO_KEY,
   ( BYTE* )ptbbutton,nButtonCount*sizeof( TBBUTTON ) );
delete []ptbbutton;
```

用ClassWizard为主框架类增加 WM_CLOSE消息处理函数，并在处理函数中调用SaveToolBars()函数。同时调用CMainFrame::SaveBarState()函数来保存应用程序中每个工具栏的当前位置和状态：

```
void CMainFrame::OnClose()
{
   // save all custom toolbars
   SaveToolbars();

   // save state of all control bars
   SaveBarState( “Control Bar States” );

   CMDIFrameWnd::OnClose();
}
```

参考本实例结尾的程序清单 —— 主框架类，可以查看SaveToolBars()的完整代码列表。

8) 为主框架类增加LoadToolbars()函数

在CMainFrame中增加新函数LoadToolbars()。首先更新将要创建的工具栏的数目，如果没有则返回空：

```
int CMainFrame::LoadToolbars()
{
   int nNumToolBars = AfxGetApp() ->
      GetProfileInt( TOOLBAR_SUMMARY_KEY, TOOLBAR_NUM_KEY,0 );
   if ( nNumToolBars )
   {
```

如果有工具栏要创建，则遍历并创建每个工具栏。由于以后将用 CMainframe :: LoadBarState ()来恢复所有控件条的状态，所以在此不必担心各工具栏的当前状态:

```
int idc = IDC_CUSTOM_TOOLBARS;
for ( int i = 0;i < nNumToolBars;i++ )
{
    // create empty toolbar
    CToolBar *pToolBar = new CToolBar;
    m_ToolBarList.AddTail( pToolBar );
    pToolBar ->
        Create( this, WS_CHILD|WS_VISIBLE|CBRS_TOP|CBRS_TOOLTIPS, idc++ );
    // set button sizes
    SIZE sizeButton, sizeImage;
    sizeImage.cx = BUTTON_WIDTH;
    sizeImage.cy = BUTTON_HEIGHT;
    sizeButton.cx = sizeImage.cx + BUTTON_XSPACING;
    sizeButton.cy = sizeImage.cy + BUTTON_YSPACING;
    pToolBar -> SetSizes( sizeButton, sizeImage );
    pToolBar -> EnableDocking( CBRS_ALIGN_ANY | CBRS_FLOAT_MULTI );
    // add all possible tool button bitmaps to this empty toolbar
    //     after first finding out how many buttons are in this bitmap
    BITMAP bm;
    CBitmap bitmap;
    bitmap.LoadBitmap( IDR_MAINFRAME );
    bitmap.GetObject( sizeof(BITMAP), &bm );
    pToolBar -> GetToolBarCtrl().AddBitmap(
        bm.bmWidth/BUTTON_WIDTH,IDR_MAINFRAME );

    // create key for this toolbar
    CString key;
    key.Format( TOOLBAR_KEY, i );
```

根据保存在系统注册表中的配置，添加按钮到这些工具栏中：

```
    // get number of buttons in this toolbar
    int nNumButtons = AfxGetApp() ->
        GetProfileInt(key,NUM_OF_BUTTONS_KEY,0);

    // get button info and insert buttons into created toolbar
    UINT k;
    TBBUTTON *ptbbutton =;
    AfxGetApp()->
        GetProfileBinary( key,BUTTON_INFO_KEY, ( BYTE ** )&ptbbutton,&k );
    for ( int j = 0;j < nNumButtons;j++ )
    {
        pToolBar -> GetToolBarCtrl().InsertButton(j,ptbbutton+j);
    }
    delete []ptbbutton;
}
```

9) 修改主框架类的OnCreate()函数

当应用程序首次启动的时候，系统注册表内没有工具栏信息，因此 LoadToolbars()将不装载工具栏而返回，应用程序将初始化为没有工具栏。因此，如果注册表内没有保存工具栏信息，可以如下编写代码，使得可以从应用程序的资源中装载缺省的工具栏：

```
// create and load custom toolbars
EnableDocking( CBRS_ALIGN_ANY );
if ( !LoadToolbars() )
{
   // if none, load standard toolbar(s)
   CToolBar *pToolBar = new CToolBar;
   if ( !pToolBar -> Create( this ) ||
      !pToolBar -> LoadToolBar( IDR_MAINFRAME ) )
   {
      TRACE0( "Failed to create toolbar\n" );
      return -1;    // fail to create
   }
:  :  :
```

在OnCreate()函数的结尾调用LoadBarState()来恢复每个工具栏的位置、大小和状态：

```
// reload all bar states
LoadBarState( "Control Bar States" );

return 0;
```

参考程序清单——主框架类，可以查看主框架类的完整代码列表。

最后一条：当用户单击了浮动工具栏的关闭按钮时会发生什么情况？缺省的行为只是隐藏该工具栏。但在本实例中将自动地删除该工具栏。一般地，可以创建自己的工具栏类并处理WM_CLOSE消息来销毁窗口。但是当工具栏正在浮动的时候，它实际上位于 Mini Dock Frame窗口上。因此需要创建自己的CMiniDockFrameWnd派生类并为其增加WM_CLOSE消息处理函数以销毁窗口。

10) 销毁浮动工具栏

用ClassWizard创建一个新的CMinFrameWnd派生类，然后用CMiniDockFrameWnd取代所有的CMiniFrameWnd。CMiniDockFrameWnd没有文档化也不被ClassWizard所支持。

用ClassWizard为该类增加WM_CLOSE消息处理函数，这里将销毁窗口而不是隐藏它：

```
void CWzdMiniDockFrameWnd::OnClose()
{
   DestroyWindow();
   CMiniDockFrameWnd::OnClose();
}
```

为了迫使应用程序使用新类，需要在 CMainFrame::OnCreate()函数的EnableDock()一行之后增加下列代码行：

```
m_pFloatingFrameClass = RUNTIME_CLASS(CWzdMiniDockFrameWnd);
```

此处不能只用自己的新类取代原来的类就算完事，这是因为 MFC用CObject:: Create-Object ()创建了该浮动微型框架。因此该方法属于非正式的方法，并没有列入 MFC文档。

当工具栏被销毁后，需要将其从应用程序包括的工具栏列表中删除。为此需使用ClassWizard创建CToolBarCtrl的一个新工具栏类。用文本编辑器将 CToolBarCtrl用CToolBar来取代。

用ClassWizard来为新工具栏类增加WM_DESTROY消息处理函数。从中访问主框架类中的工具栏列表以删除该工具栏：

```
void CWzdToolBar::OnDestroy()
{
   CToolBar::OnDestroy();
   CList<CToolBar*,CToolBar*> *pList =
      ( ( CMainFrame* )AfxGetMainWnd() ) -> GetToolBarList();
   POSITION pos = pList -> Find( this );
   if ( pos )
   {
      pList -> RemoveAt( pos );
   }

}
```

4. 注意

■ 由于实例篇幅所限，介绍的内容省略了很多精华。举个例子，当用户在属性页中选中一个工具栏位图“ 按钮 ”的时候，可以在页面上放一些描述性的消息以描述按钮。当鼠标设置为move button模式的时候，可以在工具栏上绘制一条线指示按钮将插入的位置 (当前按钮被插入到用户释放鼠标按钮的位置)。和Developer Studio的可编程工具栏一样，还可以在本实例中增加工具栏种类，一次仅显示某些特定的工具栏位图。

■ 当用户单击了浮动工具栏的关闭按钮，本实例将从应用程序的工具栏列表中删除保存在CMainFrame中的工具栏。用户也可以在应用程序的View菜单命令下维护工具栏的动态列表。

5. 使用光盘时注意

执行随书附带光盘上的工程时，单击 Option，然后单击Preferences菜单命令来调用首选项属性表单。为了从已有工具栏中删除一个按钮，可以将它拖到首选项表单。如果要将其添加回到工具栏，则从首选项表单中拖动对应的图像到工具栏即可。为了创建新的工具栏，可从任何位置拖动一个按钮到空白区域。若要删除工具栏则可以将其浮动，然后再单击其关闭按钮。

6. 程序清单——主框架类

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

#if !defined( AFX_MAINFRM_H__CA9038EA_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_ )
#define AFX_MAINFRM_H__CA9038EA_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_

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

#include "ToolBarPage.h"
#include <afxtempl.h>

class CMainFrame : public CMDIFrameWnd
{
   DECLARE_DYNAMIC( CMainFrame )
public:
   CMainFrame();

// Attributes
```

```
public:

// Operations
public:
    void LoadToolBars();
    void SaveToolBars();
    CList<CToolBar*,CToolBar*> *GetToolBarList(){return &m_ToolBarList;};

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CMainFrame )
    virtual BOOL PreCreateWindow( CREATESTRUCT& cs );
    // }}AFX_VIRTUAL
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump( CDumpContext& dc ) const;
#endif

protected:    // control bar embedded members
    CStatusBar  m_wndStatusBar;

// Generated message map functions
protected:
    // {{AFX_MSG(CMainFrame)
    afx_msg int OnCreate( LPCREATESTRUCT lpCreateStruct );
    afx_msg void OnClose();
    afx_msg void OnOptionsPreferences();
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CToolBarPage *m_pToolBarPage;

    int LoadToolbars();
    void SaveToolbars();
    CList<CToolBar*,CToolBar*> m_ToolBarList;

};

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

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

#endif
    // !defined( AFX_MAINFRM_H__CA9038EA_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_ )
// MainFrm.cpp : implementation of the CMainFrame class
//
```

```
#include "stdafx.h"
#include "Wzd.h"
#include "WzdProject.h"
#include "WzdToolBar.h"
#include "WzdMiniDockFrameWnd.h"

#include "MainFrm.h"

#include <afxpriv.h>

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

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

IMPLEMENT_DYNAMIC( CMainFrame, CMDIFrameWnd )

BEGIN_MESSAGE_MAP( CMainFrame, CMDIFrameWnd )
    // {{AFX_MSG_MAP( CMainFrame )
    ON_WM_CREATE()
    ON_WM_CLOSE()
    ON_COMMAND( ID_OPTIONS_PREFERENCES, OnOptionsPreferences )
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,          // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

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

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
    while ( !m_ToolBarList.IsEmpty() )
    {
```

```
        delete m_ToolBarList.RemoveHead();
    }
}

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

    // create and load custom toolbars
    EnableDocking( CBRS_ALIGN_ANY );
    m_pFloatingFrameClass = RUNTIME_CLASS( CWzdMiniDockFrameWnd );
    if ( !LoadToolbars() )
    {
        // if none, load standard toolbar(s)
        CToolBar *pToolBar = ( CWzdToolBar* )new CWzdToolBar;
        if ( !pToolBar -> Create( this ) ||
            !pToolBar -> LoadToolBar( IDR_MAINFRAME ) )
        {
            TRACE0( "Failed to create toolbar\n" );
            return -1;    // fail to create
        }
        m_ToolBarList.AddTail( pToolBar );

        // TODO: Remove this if you don't want tool tips or a resizeable toolbar
        pToolBar -> SetBarStyle( pToolBar -> GetBarStyle() |
            CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC );

        // TODO: Delete these three lines if you don't want the toolbar to
        //     be dockable
        pToolBar -> EnableDocking( CBRS_ALIGN_ANY );
        DockControlBar( pToolBar );
    }

    if ( !m_wndStatusBar.Create( this ) ||
        !m_wndStatusBar.SetIndicators( indicators,
        sizeof( indicators )/sizeof( UINT ) ) )
    {
        TRACE0( "Failed to create status bar\n" );
        return -1;    // fail to create
    }

    // reload all bar states
    LoadBarState( "Control Bar States" );
    return 0;
}

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

```
    return CMDIFrameWnd::PreCreateWindow( cs );
}

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

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

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

#endif    //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnClose()
{
    // save all custom toolbars
    SaveToolbars();

    // save state of all control bars
    SaveBarState( “Control Bar States” );

    CMDIFrameWnd::OnClose();
}

void CMainFrame::OnOptionsPreferences()
{
    CPropertySheet sheet( _T( "Preferences" ),this );
    m_pToolBarPage = new CToolBarPage;
    sheet.AddPage( m_pToolBarPage );
    sheet.DoModal();
    delete m_pToolBarPage;
}

int CMainFrame::LoadToolbars()
{
    int nNumToolBars = AfxGetApp() ->
        GetProfileInt( TOOLBAR_SUMMARY_KEY,TOOLBAR_NUM_KEY,0 );
    if ( nNumToolBars )
    {
        int idc = IDC_CUSTOM_TOOLBARS;
        for ( int i = 0;i < nNumToolBars;i++ )
        {
```

```
            // create empty toolbar
            CToolBar *pToolBar = ( CToolBar* )new CWzdToolBar;
            m_ToolBarList.AddTail( pToolBar );
            pToolBar ->
               Create( this, WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_TOOLTIPS, idc++ );
            SIZE sizeButton, sizeImage;
            sizeImage.cx = BUTTON_WIDTH;
            sizeImage.cy = BUTTON_HEIGHT;
            sizeButton.cx = sizeImage.cx + BUTTON_XSPACING;
            sizeButton.cy = sizeImage.cy + BUTTON_YSPACING;
            pToolBar -> SetSizes( sizeButton, sizeImage );
            pToolBar -> EnableDocking( CBRS_ALIGN_ANY | CBRS_FLOAT_MULTI );

            // add all possible tool button bitmaps to this empty toolbar
            //    after first finding out how many buttons are in this bitmap
            BITMAP bm;
            CBitmap bitmap;
            bitmap.LoadBitmap( IDR_MAINFRAME );
            bitmap.GetObject( sizeof(BITMAP), &bm );
            pToolBar ->
               GetToolBarCtrl().AddBitmap( bm.bmWidth/BUTTON_WIDTH,IDR_MAINFRAME );

            // create key for this toolbar
            CString key;
            key.Format( TOOLBAR_KEY,i );

            // get number of buttons in this toolbar
            int nNumButtons = AfxGetApp() ->
               GetProfileInt( key,NUM_OF_BUTTONS_KEY,0 );

            // get button info and insert buttons into created toolbar
            UINT k;
            TBBUTTON *ptbbutton;
            AfxGetApp() ->
               GetProfileBinary( key,BUTTON_INFO_KEY,( BYTE ** )&ptbbutton,&k );
            for ( int j = 0;j < nNumButtons;j++ )
            {
               pToolBar -> GetToolBarCtrl().InsertButton( j,ptbbutton+j );
            }
            delete []ptbbutton;
         }
      }
   return(nNumToolBars);
   }

   void CMainFrame::SaveToolbars()
   {
      int nNumToolBars = m_ToolBarList.GetCount();
      AfxGetApp() ->
         WriteProfileInt( TOOLBAR_SUMMARY_KEY,TOOLBAR_NUM_KEY,nNumToolBars );
      if ( nNumToolBars )
```

```
    {
        int i = 0;
        int idc = IDC_CUSTOM_TOOLBARS;
        for ( POSITION pos = m_ToolBarList.GetHeadPosition();pos;i++ )
        {
            CToolBar *pToolBar = m_ToolBarList.GetNext( pos );

            // create key for this toolbar
            CString key;
            key.Format( TOOLBAR_KEY,i );

            // give toolbar a unique, sequential ID for SaveBarState/LoadBarState
            pToolBar -> SetDlgCtrlID( idc++ );

            // write number of buttons in this toolbar
            int nButtonCount = pToolBar -> GetToolBarCtrl().GetButtonCount();
            AfxGetApp() -> WriteProfileInt( key,NUM_OF_BUTTONS_KEY,nButtonCount );

            // write info on each button
            TBBUTTON *ptbbutton = new TBBUTTON[nButtonCount];
            for ( int j = 0;j < nButtonCount;j++ )
            {
                pToolBar -> GetToolBarCtrl().GetButton( j,ptbbutton+j );
            }
            AfxGetApp() ->
                WriteProfileBinary( key,BUTTON_INFO_KEY,
                ( BYTE* )ptbbutton,nButtonCount*sizeof( TBBUTTON ) );
            delete []ptbbutton;
        }
    }
}
```

7. 程序清单 —— 属性页类

```
#if !defined TOOLBARPAGE_H
#define TOOLBARPAGE_H

// ToolBarPage.h : header file
//

#include “WzdProject.h”

/////////////////////////////////////////////////////////////////////////////
// CToolBarPage dialog

class CToolBarPage : public CPropertyPage
{
    DECLARE_DYNCREATE( CToolBarPage )

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

```
    ~CToolBarPage();

// Dialog Data
    // {{AFX_DATA( CToolBarPage )
    enum { IDD = IDD_TOOLBAR_PAGE };
    // }}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    // {{AFX_VIRTUAL( CToolBarPage )
public:
    virtual BOOL OnSetActive();
    virtual BOOL OnKillActive();
protected:
    virtual void DoDataExchange( CDataExchange* pDX );    // DDX/DDV support
    // }}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    // {{AFX_MSG( CToolBarPage )
    virtual BOOL OnInitDialog();
    afx_msg void OnLButtonDown( UINT nFlags, CPoint point );
    afx_msg void OnLButtonUp( UINT nFlags, CPoint point );
    afx_msg void OnPaint();
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CToolBar        m_ToolBar;
    CBitmap         m_bitmap;
    int             m_xBitmapStart;
    int             m_yBitmapStart;
    int             m_nButtonCount;

    BOOL            m_bMoving;
    int             m_nButtonMoved;
    TBBUTTON        m_tbbutton;
    HICON           m_hMoveCursor;
    CToolBar        *m_pToolBar;
    CToolBar        *GetToolBar( CPoint point );
    int             GetButtonIndex( CToolBar *pToolBar, CPoint point );
    void            UpdateToolBar( CToolBar *pToolBar );
    BOOL            GetToolBarNear( CPoint &point,UINT &nMode,BOOL &bHorz );

};

#endif
// ToolBarPage.cpp : implementation file
//

#include “stdafx.h”
```

```
#include "wzd.h"
#include "ToolBarPage.h"
#include "WzdToolBar.h"
#include "WzdProject.h"
#include "MainFrm.h"

#include <afxcmn.h>
#include <afxpriv.h>

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


/////////////////////////////////////////////////////////////////////////////
// CToolBarPage property page

IMPLEMENT_DYNCREATE( CToolBarPage, CPropertyPage )

CToolBarPage::CToolBarPage() : CPropertyPage( CToolBarPage::IDD )
{
   // {{AFX_DATA_INIT( CToolBarPage )
   // }}AFX_DATA_INIT
   m_bMoving = FALSE;
   m_nButtonCount = 0;
   m_pToolBar = NULL;
   m_hMoveCursor = AfxGetApp() -> LoadCursor( IDI_MOVING_BUTTON );
}

CToolBarPage::~CToolBarPage()
{
}

void CToolBarPage::DoDataExchange( CDataExchange* pDX )
{
   CPropertyPage::DoDataExchange( pDX );
   // {{AFX_DATA_MAP( CToolBarPage )
   // }}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP( CToolBarPage, CPropertyPage )
   // {{AFX_MSG_MAP(CToolBarPage)
   ON_WM_LBUTTONDOWN()
   ON_WM_LBUTTONUP()
   ON_WM_PAINT()
   // }}AFX_MSG_MAP
END_MESSAGE_MAP()

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

```
// CToolBarPage message handlers

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

    // create and load breeder toolbar
    m_ToolBar.Create( this );
    m_ToolBar.LoadToolBar( IDR_MAINFRAME );
    // load breeder toolbar bitmap
    m_bitmap.LoadBitmap( IDR_MAINFRAME );

    // get window and button count
    m_nButtonCount = m_ToolBar.GetToolBarCtrl().GetButtonCount();


    CRect rect;
    GetClientRect( &rect );
    m_xBitmapStart = rect.Width()/6;
    m_yBitmapStart = rect.Height()/6;

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

void CToolBarPage::OnPaint()
{
    CPaintDC dc( this );    // device context for painting

    // display breeder toolbar bitmap
    CDC memDC;
    memDC.CreateCompatibleDC( &dc );
    memDC.SelectObject( &m_bitmap );

    int x = m_xBitmapStart;
    int y = m_yBitmapStart;
    for ( int i = 0; i < m_nButtonCount; i++ )
    {
        dc.BitBlt( x,y,BUTTON_WIDTH,BUTTON_HEIGHT, &memDC, i*BUTTON_WIDTH,
            0, SRCCOPY );
        x += BUTTON_WIDTH + BUTTON_XSPACING;
    }
    memDC.DeleteDC();
}

BOOL CToolBarPage::OnSetActive()
{
    SetCapture();
    return CPropertyPage::OnSetActive();
}

BOOL CToolBarPage::OnKillActive()
```

```
{
    ReleaseCapture();
    return CPropertyPage::OnKillActive();
}

void CToolBarPage::OnLButtonDown( UINT nFlags, CPoint point )
{
    m_bMoving = FALSE;
    ClientToScreen( &point );
    CWnd *pWnd = WindowFromPoint( point );

    // if point is parent or child of
    if ( pWnd != this && ( pWnd == GetParent() ||
        GetParent() -> IsChild( pWnd ) ) )
    {
        CPoint pt( point );
        pWnd -> ScreenToClient( &pt );
        if ( pt.y >= 0 )
        {
            pWnd -> SendMessage( WM_LBUTTONDOWN,nFlags,MAKELONG( pt.x,pt.y ) );
        }
        else
        {
            UINT ht = pWnd ->
                SendMessage( WM_NCHITTEST,0,MAKELONG( point.x,point.y ) );
            pWnd ->
                SendMessage( WM_NCLBUTTONDOWN,ht,MAKELONG( point.x,point.y ) );
        }
        return;
    }

    // see if this is a toolbar button
    if ( m_pToolBar = GetToolBar( point ) )
    {

        m_nButtonMoved = GetButtonIndex( m_pToolBar, point );
        m_pToolBar -> GetToolBarCtrl().GetButton( m_nButtonMoved,&m_tbbutton );
        m_bMoving = TRUE;
        ::SetCursor( m_hMoveCursor );
    }

    // else if this window
    else if ( pWnd == this )
    {
        ScreenToClient( &point );
        point.Offset( -m_xBitmapStart,-m_yBitmapStart );
        CRect rect( 0,0,
            ( BUTTON_WIDTH+BUTTON_XSPACING )*m_nButtonCount,BUTTON_HEIGHT );
        if ( rect.PtInRect( point ) )
        {
            m_nButtonMoved = 0;
            int i = point.x/( BUTTON_WIDTH + BUTTON_XSPACING );
```

```
            for ( int j = 0;j < m_nButtonCount;j++ )
            {
                UINT k;
                int l;
                m_ToolBar.GetButtonInfo( j,k,k,l );
                if ( l == i )
                {
                    m_nButtonMoved = j;
                    break;
                }
            }
            m_ToolBar.GetToolBarCtrl().GetButton( m_nButtonMoved,&m_tbbutton );
            m_bMoving = TRUE;
            ::SetCursor( m_hMoveCursor );
            SetCapture();
        }
    }

    CPropertyPage::OnLButtonDown( nFlags, point );
}

void CToolBarPage::OnLButtonUp( UINT nFlags, CPoint point )
{
    if ( m_bMoving )
    {

        // delete button from source toolbar
        if ( m_pToolBar )
        {
            m_pToolBar -> GetToolBarCtrl().DeleteButton( m_nButtonMoved );
        }

        // if dropped anywhere but toolbar property page
        CRect rect;
        ClientToScreen( &point );
        GetWindowRect( &rect );
        if ( !rect.PtInRect( point ) )
        {
            // if dropped on existing toolbar, add button to it
            CToolBar *pToolBar;
            if ( pToolBar = GetToolBar( point ) )
            {
                int i = GetButtonIndex( pToolBar,point );
                pToolBar -> GetToolBarCtrl().InsertButton( i,&m_tbbutton );
                UpdateToolBar( pToolBar );
            }
            // else create a new toolbar and add our button to it
            else
            {
                pToolBar = ( CWzdToolBar* )new CWzdToolBar;
                CList<CToolBar*,CToolBar*> *pList =
                    ( ( CMainFrame* )AfxGetMainWnd() ) -> GetToolBarList();
```

```
            pList -> AddTail( pToolBar );
            pToolBar ->
               Create( GetParentFrame(),
               WS_CHILD|WS_VISIBLE|CBRS_TOP|CBRS_TOOLTIPS );
            SIZE sizeButton, sizeImage;
            sizeImage.cx = BUTTON_WIDTH;
            sizeImage.cy = BUTTON_HEIGHT;
            sizeButton.cx = sizeImage.cx + BUTTON_XSPACING;
            sizeButton.cy = sizeImage.cy + BUTTON_YSPACING;
            pToolBar -> SetSizes( sizeButton, sizeImage );
            pToolBar -> EnableDocking( CBRS_ALIGN_ANY | CBRS_FLOAT_MULTI );

            // add all possible tool button bitmaps
            pToolBar ->
               GetToolBarCtrl().AddBitmap( m_nButtonCount,IDR_MAINFRAME );

            // add new button to this new toolbar
            pToolBar -> GetToolBarCtrl().InsertButton( 0,&m_tbbutton );

            UINT nMode;
            BOOL bHorz;
            if ( GetToolBarNear( point,nMode,bHorz ) )
            {
               // dock toolbar centered on cursor
               CSize size = pToolBar -> CalcFixedLayout( FALSE,bHorz );
               if ( bHorz )
                  point.x -= size.cx/2;
               else
                  point.y -= size.cy/2;
               CRect rectx( point,size );
               GetParentFrame() -> DockControlBar( pToolBar,nMode,&rectx );
               pToolBar -> Invalidate();
               GetParentFrame() -> RecalcLayout();
            }
            else
            {
               GetParentFrame() ->
                  FloatControlBar( pToolBar, point, CBRS_ALIGN_TOP );
            }
         }
      }
      // else update source toolbar if any
      else if ( m_pToolBar )
      {
         UpdateToolBar( m_pToolBar );

         // if no buttons left on toolbar, delete it
         if ( m_pToolBar && m_pToolBar ->
            GetToolBarCtrl().GetButtonCount() == 0 )
         {
            POSITION pos;
            CList<CToolBar*,CToolBar*> *pList =
```

```
            ( ( CMainFrame* )AfxGetMainWnd() ) -> GetToolBarList();
        if ( pos = pList -> Find( m_pToolBar ) )
        {
            pList -> RemoveAt( pos );
            m_pToolBar -> m_bAutoDelete = TRUE;
            if ( !m_pToolBar -> IsFloating() )
            {
                m_pToolBar -> SendMessage( WM_CLOSE );
            }
            else
            {
                CDockBar* pDockBar = m_pToolBar -> m_pDockBar;
                CMiniDockFrameWnd* pDockFrame =
                    ( CMiniDockFrameWnd* )pDockBar -> GetParent();
                pDockFrame -> ShowWindow( SW_HIDE );
                pDockFrame -> SendMessage( WM_CLOSE );
            }
        }
      }
      m_bMoving = FALSE;
      ::SetCursor( AfxGetApp() -> LoadStandardCursor( IDC_ARROW ) );
   }

   CPropertyPage::OnLButtonUp( nFlags, point );
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// Helper Toolbar Functions

CToolBar *CToolBarPage::GetToolBar( CPoint point )
{
   CRect rect;
   CList<CToolBar*,CToolBar*> *pList =
      ( ( CMainFrame* )AfxGetMainWnd() ) -> GetToolBarList();
   if ( !pList -> IsEmpty() )
   {
      for ( POSITION pos = pList -> GetHeadPosition(); pos; )
      {
         CToolBar *pToolBar = pList -> GetNext( pos );
         pToolBar -> GetWindowRect( &rect );
         if ( rect.PtInRect( point ) )
         {
            return pToolBar;
         }
      }
   }
   return( NULL );
}

int CToolBarPage::GetButtonIndex( CToolBar *pToolBar, CPoint point )
```

```
{
    CRect rect;
    pToolBar -> ScreenToClient( &point );
    int nButtons = pToolBar ->GetToolBarCtrl().GetButtonCount();
    for ( int i = 0; i < nButtons; i++ )
    {
        pToolBar -> GetItemRect( i,&rect );
        if ( rect.PtInRect( point ) )
        {
            return( i );
        }
    }
    return( -1 );
}

void CToolBarPage::UpdateToolBar( CToolBar *pToolBar )
{
    if ( pToolBar )
    {
        if ( pToolBar -> IsFloating() )
        {
            CDockBar* pDockBar = pToolBar -> m_pDockBar;
            CMiniDockFrameWnd* pDockFrame =
                ( CMiniDockFrameWnd* )pDockBar -> GetParent();
            pDockFrame -> RecalcLayout( TRUE );
            pDockFrame -> UpdateWindow();
        }
        else
        {
            pToolBar -> Invalidate();
            GetParentFrame() -> RecalcLayout();
        }
    }
}

BOOL CToolBarPage::GetToolBarNear( CPoint &point,UINT &nMode,BOOL &bHorz )
{
    CRect rect;
    BOOL bDock = FALSE;
    CDockState dockstate;
    ( ( CMainFrame * )AfxGetMainWnd() ) -> GetDockState( dockstate );
    for ( int i = 0; i < dockstate.m_arrBarInfo.GetSize(); i++ )
    {
        CControlBarInfo *pBarInfo =
            ( CControlBarInfo * )dockstate.m_arrBarInfo[i];
        CControlBar *pBar = ( CControlBar * )pBarInfo -> m_pBar;
        // if this bar is visible, can be docked and isn't floating, check it out
        if ( pBarInfo - >m_bVisible && pBarInfo ->
            m_bDocking && !pBar -> IsFloating() )
        {
            pBarInfo -> m_pBar -> GetWindowRect( &rect );
```

```
            DWORD dwStyle = pBar -> GetBarStyle();
            if ( bHorz = ( dwStyle & CBRS_ORIENT_HORZ ) )
            {
               // if user clicked in this region, dock new toolbar here
               if ( point.y >= rect.top && point.y < rect.bottom)
               {
                  bDock = TRUE;
                  point.y = rect.top;
                  if ( dwStyle & CBRS_ALIGN_TOP )
                     nMode = AFX_IDW_DOCKBAR_TOP;
                  else
                     nMode = AFX_IDW_DOCKBAR_BOTTOM;
                  break;
               }
            }
            else
            {
               // else if user clicked in this region, dock here
               if ( point.x >= rect.left && point.x < rect.right )
               {
                  bDock = TRUE;
                  point.x = rect.left;
                  if ( dwStyle & CBRS_ALIGN_LEFT )
                     nMode = AFX_IDW_DOCKBAR_LEFT;
                  else
                     nMode = AFX_IDW_DOCKBAR_RIGHT;
                  break;
               }
            }
         }
      }
      return( bDock );
   }
```

5.4 实例10：在对话框中添加工具栏、菜单和状态栏

1. 目标

如图5-4所示为对话框应用程序增加菜单、工具栏和状态栏。

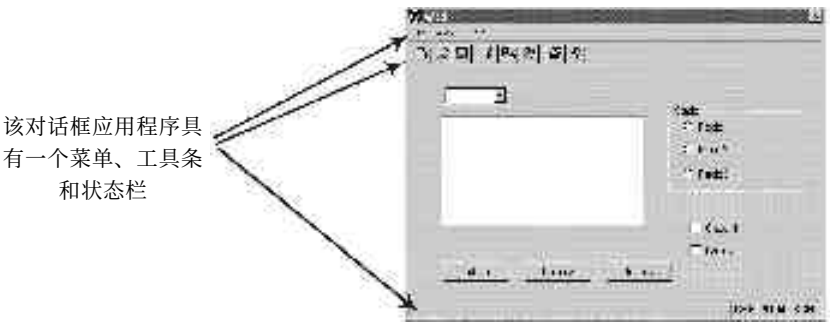

图5-4 具有工具栏和状态栏的对话框

2. 策略

与SDI和MDI应用程序不同，AppWizard不能创建具有菜单、工具栏和状态栏的对话框应用程序。这并不意味着对话框应用程序就不能具有这些用户输入方式，毕竟所有具有菜单、工具栏或者状态栏的弹出窗口实际上都是可以添加到其他窗口的控件窗口，不同之处在于需要手工添加这些项目。

3. 步骤

1) 增加菜单

用Menu Editor创建新菜单，或者从另一个工程中拷贝一个。甚至可以创建一个新的 SDI应用程序，并从其.rc文件中获取器菜单并将其粘贴到本对话框应用程序的 .rc文件。

用Dialog Editor将该菜单增加到应用程序的对话框模板。

用ClassWizard为对话框类增加命令消息处理函数以处理这些菜单项。

2) 增加工具栏

用Toolbar Editor创建新工具栏，或者如上所述的最后步骤从另一个工程中拷贝。

由于Dialog Editor不允许在模板中添加工具栏控件，所以需要用自己的对话框类动态地创建一个。确定创建该工具栏的位置的一个好办法是用 Dialog Editor在要添加工具栏的地方创建一个静态控件占位符 (placeholder)，然后将该控件的位置和大小传送给创建中的工具栏。因此用Dialog Editor来增加静态控件到要添加工具栏的对话框模板，将可以使工具栏看起来又细又长。

在对话框类中嵌入CToolbar变量：

```
CToobar  m_wndToolBar;
```

在对话框类的OnInitDialog()消息处理函数内创建工具栏：

```
if ( !m_wndToolBar.Create( this ) ||
   !m_wndToolBar.LoadToolBar( IDR_MAINFRAME ) )
{
   TRACE0( "Failed to create toolbar\n" );
   return -1;    // fail to create
}
```

在先前用对话框编辑器创建的静态控件内定位工具栏：

```
CRect rect;
GetDlgItem( IDC_TOOLBAR_STATIC ) -> GetWindowRect( &rect );
ScreenToClient( &rect );
m_wndToolBar.MoveWindow( &rect );
```

3) 增加状态栏

用String Table Editor定义状态栏窗格：

```
ID_INDICATOR_ONE    "xxx"
ID_INDICATOR_TWO    "yyy"
```

再次使用Dialog Editor向对话框模板中将要添加状态栏的位置上添加静态控件。

在对话框类中嵌入一个状态栏控件：

```
CStatusBar   m_wndStatusBar;
```

将用String Editor创建的状态窗格列表增加到对话框类代码的顶部：

```
static UINT indicators[] =
```

```
{
    ID_SEPARATOR,              // status line indicator
    ID_INDICATOR_ONE,
    ID_INDICATOR_TWO,
};
```

创建状态栏并将其定位于静态控件之上：

```
if ( !m_wndStatusBar.Create(this, WS_CHILD|WS_VISIBLE|WS_BORDER) ||
    !m_wndStatusBar.SetIndicators( indicators,
    sizeof( indicators )/sizeof( UINT ) ) )
{
    TRACE0( "Failed to create status bar\n" );
    return -1;    // fail to create
}

GetDlgItem( IDC_STATUSBAR_STATIC ) -> GetWindowRect( &rect );
ScreenToClient( &rect );
m_wndStatusBar.MoveWindow( &rect );
```

4. 注意

■ 由CStatusBar创建的状态栏中的第一个窗格没有边界。虽然这对 SDI或者MDI应用程序没有什么影响，因为状态栏一般是位于视图的下方，但在对话框应用程序中该窗格实际上却消失在对话框窗口内了。为了使该窗格具有明确的边界可以在状态栏创建后使用下列代码：

```
UINT nID, nStyle;
int nWidth;
m_wndStatusBar.GetPaneInfo( 0, nID, nStyle, nWidth );
m_wndStatusBar.SetPaneInfo( 0, nID, SBPS_STRETCH, nWidth );
```

■ 正如工具栏和状态栏必须手工增加一样，程序员还要负责手工更新它们的状态。对工具栏就意味手工变灰或者检查各个按钮，对状态栏则意味着手工设置每个窗格的文本。

■ 如果已经实现菜单，可以从对话框应用程序的系统菜单中删除帮助菜单项，然后在对话框类中删除其支持逻辑代码。

5. 使用光盘时注意

执行随书附带光盘上的工程的时候，将会注意到该对话框应用程序有一个菜单、工具栏和状态栏。

5.5 实例11：在弹出菜单中增加位图标记

1. 目标

如图5-5所示在弹出菜单中增加位图标记。

2. 策略

实际上只对位图使用一点小技巧即可。并不需要为菜单增加一个位图，而是打开具有垂直而细长的位图对话框，在与之相邻处可以打开一个弹出菜单。另一个更复杂的方式将涉及到自己绘制

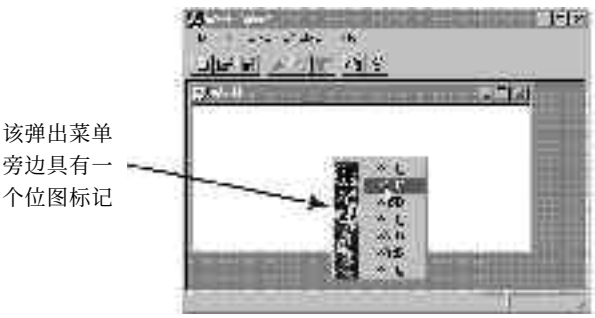

图5-5 具有标记的弹出菜单

菜单，这可以参考实例7。

3. 步骤

1) 创建弹出菜单对话框类

为所需的菜单项创建弹出菜单资源。

创建一个对话框资源，其一边是图像控件，另一边是静态控件。配置该图像控件使其拥有出现在静态控件旁边的位图。

重新改变静态控件和对话框模板的大小，使它们与弹出菜单的大小相符合。由于还需要完成以下步骤才能看见最后结果，因此目前只能进行大概的估计。

用ClassWizard创建一个使用该模板的对话框类。

用ClassWizard为该对话框类增加WM_INITDIALOG消息处理函数。在此可以查找光标位置，重新定位对话框以便于模仿菜单出现在光标旁边的感觉:

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

// position where mouse button was clicked
CPoint pt;
GetCursorPos( &pt );
SetWindowPos( NULL,pt.x,pt.y,0,0,SWP_NOSIZE );

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

用ClassWizard为该类增加WM_PAINT消息处理函数，用于在静态控件之上打开弹出菜单:

```
void CWzdDlg::OnPaint()
{
   CDialog::OnPaint();

   // load up menu
   CMenu menu;
   menu.LoadMenu( IDR_WZD_MENU );
   CMenu* pPopup = menu.GetSubMenu( 0 );

   // get location of static control and display popup menu there
   CRect rect;
   GetDlgItem( IDC_MENU_STATIC ) -> GetWindowRect( &rect );
   int nLeft = rect.right + 2;
   GetWindowRect( &rect );
   pPopup ->
      TrackPopupMenu( TPM_RIGHTBUTTON, nLeft, rect.top, GetParent() );

   // cancel this dialog
   PostMessage( WM_CLOSE );
}
```

注意一旦用户选择菜单项，将发送一个消息给自己以便立即关闭该对话框。

2) 实现弹出菜单对话框类

用ClassWizard向欲实现该菜单的类中增加 WM_RBUTTONDOWN消息处理函数。在该实例中，该菜单用View类实现。这里将用DoModal()创建对话框：

```
void CWzdView::OnRButtonDown( UINT nFlags, CPoint point )
{
   CWzdDlg dlg;
   dlg.DoModal();

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

4. 注意

正如以上提到过的，还可以自己绘制菜单，将菜单项文本定位于居左或者居右的位置，并且在绘制菜单项时绘制位图的某些部分，这种方式也可以实现以上的效果。

5. 使用光盘时注意

执行随书附带光盘上的工程时，在视图内用鼠标右键单击并打开弹出菜单，将在其旁边显示位图。

5.6 实例12：工具栏上的下拉按钮

1. 目标

如图5-6所示在工具栏上创建下拉按钮。

2. 策略

这里要用到MFC 6.0中新增加的两种工具栏风格。应用程序必须运行于Windows 98或者带有Internet Explorer 4.01 以及安装更高版本的系统上。第一种风格是TBSTYLE_ DROPDOWN风格，该风格导致工具栏按钮在其被按下时发出TBN_DROPDOWN通知消息(一般地，工具栏按钮在这种情况发出包含该按钮ID号的WM_COMMAND消息)。该通知消息被用于在按钮下方显示弹出菜单，就和一般菜单一样。第二种风格 TBSTYLE_EX_DRAWDDARROWS则导致具有第一种风格的按钮绘制出图 5-6所示的下拉箭头。

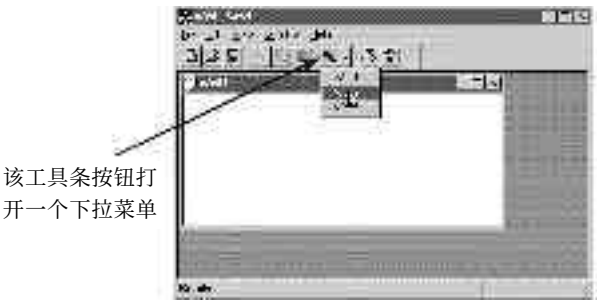

图5-6 工具栏上的下拉按钮

为了防止CMainFrame类因为应用这些新风格的代码而变得混乱，这里将其封装到用户自己的CToolbar派生工具栏类。

3. 步骤

1) 创建新的工具栏类

用ClassWizard创建CToolbarCtrl的派生类，然后用 Text Editor将所有CToolbarCtrl改变为CToolbar。

为该类增加一个新函数以装载工具栏，但是它将对选中的工具栏按钮应用两种新的工具栏风格，这将使它显示下拉按钮，并在被按下时发出 TBN_DROPDOWN通知消息：

```
BOOL CWzdToolBar::LoadToolBarEx( UINT id )
{
   // load toolbar info
   BOOL bRet;
   bRet = CToolBar::LoadToolBar( id );

   // find where our dropdown button will go
   int pos = CommandToIndex( IDC_WZD_DROPBUTTON );

   // set this button to be a dropdown button
   int iImage;
   UINT nID,nStyle;
   GetButtonInfo( pos,nID,nStyle,iImage );
   SetButtonInfo( pos,nID,nStyle|TBSTYLE_DROPDOWN,iImage );

   // ask toolbar to draw a down arrow next to this and
   //     any button with TBSTYLE_DROPDOWN
   SendMessage( TB_SETEXTENDEDSTYLE,0,TBSTYLE_EX_DRAWDDARROWS );
   return bRet;
}
```

注意需要用新工具栏消息 TB_SETEXTENDEDSTYLE来设置TBSTYLE_EX_DRAWDDARROWS风格。

如果工具栏的下拉按钮位于浮动条上，则工具栏末端的按钮可能被剪切。因为 CToolbar的CalcDynamicLayout()函数没有注意到这一事实：下拉按钮比正常的按钮要长。为了弥补这一缺陷，需要重载工具栏类的 CalcDynamicLayout()函数以加大工具栏的像素尺寸值，所以在代码中声明该重载并将其实现：

```
// in your toolbar class's declaration file:
CSize CalcDynamicLayout( int nLength, DWORD dwMode );
:   :   :
// in your toolbar class's implementation file:
CSize CWzdToolBar::CalcDynamicLayout( int nLength, DWORD dwMode )
{
   CSize size = CToolBar::CalcDynamicLayout( nLength,dwMode );
   // make room on non-fixed toolbars for additional down arrows
   if ( dwMode&LM_HORZ ) size.cx += 16*1;        // 16 * number of buttons
                                                 //     with arrows
   return size;
}
```

这一步为该类手工增加TBN_DROPDOWN消息处理函数。在声明文件内增加如下代码：

```
// {{AFX_MSG( CWzdToolBar )
// }}AFX_MSG
// add the following
void OnDropdownButton( LPNMTOOLBAR pNotifyStruct, LRESULT* result );
DECLARE_MESSAGE_MAP()
```

为消息映射增加如下代码：

```
BEGIN_MESSAGE_MAP( CWzdToolBar, CToolBar )
   // {{AFX_MSG_MAP( CWzdToolBar )
```

```
    // }}AFX_MSG_MAP
    ON_NOTIFY_REFLECT( TBN_DROPDOWN,OnDropdownButton ) // <<< add
END_MESSAGE_MAP()
```

通过在工具栏按钮右下方打开弹出菜单来处理该通知消息：

```
void CWzdToolBar::OnDropdownButton( LPNMTOOLBAR lpnmtb,
    LRESULT *result )
{
    // get location of clicked button
    CRect rect;
    GetItemRect( CommandToIndex( lpnmtb -> iItem ),&rect );
    ClientToScreen( &rect );

    // putup a popup menu there
    CMenu menu;
    menu.LoadMenu( IDR_WZD_MENU );
    CMenu* pPopup = menu.GetSubMenu( 0 );

    pPopup ->
        TrackPopupMenu( TPM_RIGHTBUTTON, rect.left, rect.bottom, this );

    *result = TBDDRET_DEFAULT;    // drop-down was handled, also
    // TBDDRET_NODEFAULT drop-down was not handled.
    // TBDDRET_TREATPRESSED generate a WM_COMMAND message
}
```

为查看该类的完整程序，可以参考程序清单 —— 工具栏类

2) 实现新的工具栏类

在CMainFrame中使用新的工具栏类并用LoadBitmapEx()装载工具栏资源。

4. 注意

如果工具栏为固定大小，则不需要考虑重载 CalcDynamicLayout()以设置工具栏大小，工具栏将总是保持其固有大小。请参阅第 2章以了解工具栏和控件条的更多内容。如果工具栏可以垂直停靠，可以注意到下拉箭头将被工具栏剪切。这可以在 CalcDynamicLayout()中修改代码以改变这一效果，但是这看起来好像是微软的特色而不像是一个程序错误 (换句话说，在其他微软应用程序中的下拉工具栏按钮也剪切下拉箭头。)

5. 使用光盘时注意

■ 执行随书附带光盘上的工程时，可以注意到工具栏上的下拉按钮，当其被单击的时候将在工具栏按钮下出现一个菜单。

■ 由于该工程需要用到只在Developer Studio v6.0中才提供的功能，所以该光盘中只有该工程的VC 6.0版本。

6. 程序清单 —— 工具栏类

```
#if !defined( AFX_WZDTOOLBAR_H__27649E31_C807_11D1_9B5D_00AA003D8695__INCLUDED_ )
#define AFX_WZDTOOLBAR_H__27649E31_C807_11D1_9B5D_00AA003D8695__INCLUDED_

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

```
// WzdToolBar.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar window

class CWzdToolBar : public CToolBar
{
// Construction
public:
    CWzdToolBar();

    BOOL LoadToolBarEx( UINT id );

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdToolBar )
    // }}AFX_VIRTUAL
    CSize CalcDynamicLayout( int nLength, DWORD dwMode );

// Implementation
public:
    virtual ~CWzdToolBar();

    // Generated message map functions
protected:
    // {{AFX_MSG( CWzdToolBar )
    // }}AFX_MSG
    void OnDropdownButton( LPNMTOOLBAR pNotifyStruct, LRESULT* result );
    DECLARE_MESSAGE_MAP()
};

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

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

#endif
    // !defined( AFX_WZDTOOLBAR_H__27649E31_C807_11D1_9B5D_00AA003D8695__INCLUDED_ )
// WzdToolBar.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
```

```
#include "WzdTlBar.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar

CWzdToolBar::CWzdToolBar()
{
}

CWzdToolBar::~CWzdToolBar()
{
}


BEGIN_MESSAGE_MAP( CWzdToolBar, CToolBar )
    // {{AFX_MSG_MAP( CWzdToolBar )
    // }}AFX_MSG_MAP
    ON_NOTIFY_REFLECT( TBN_DROPDOWN,OnDropdownButton )
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar message handlers

BOOL CWzdToolBar::LoadToolBarEx( UINT id )
{
    // load toolbar info
    BOOL bRet;
    bRet = CToolBar::LoadToolBar( id );
    // find where our dropdown button will go
    int pos = CommandToIndex( IDC_WZD_DROPBUTTON );

    // set this button to be a dropdown button
    int iImage;
    UINT nID,nStyle;
    GetButtonInfo( pos,nID,nStyle,iImage );
    SetButtonInfo( pos,nID,nStyle|TBSTYLE_DROPDOWN,iImage );

    // ask toolbar to draw a down arrow next to this and any button with
    //     TBSTYLE_DROPDOWN
    SendMessage( TB_SETEXTENDEDSTYLE,0,TBSTYLE_EX_DRAWDDARROWS );

    return bRet;
}

CSize CWzdToolBar::CalcDynamicLayout( int nLength, DWORD dwMode )
```

```
{
    CSize size = CToolBar::CalcDynamicLayout( nLength,dwMode );
    // make room on non-fixed toolbars for additional down arrows
    if (dwMode&LM_HORZ) size.cx += 16*1;    // 16 * number of buttons
                                            //     with arrows
    return size;
}

void CWzdToolBar::OnDropdownButton( LPNMTOOLBAR lpnmtb, LRESULT *result )
{
    // get location of clicked button
    CRect rect;
    GetItemRect( CommandToIndex( lpnmtb -> iItem ),&rect );
    ClientToScreen( &rect );

    // putup a popup menu there
    CMenu menu;
    menu.LoadMenu( IDR_WZD_MENU );
    CMenu* pPopup = menu.GetSubMenu( 0 );
    pPopup -> TrackPopupMenu( TPM_RIGHTBUTTON, rect.left, rect.bottom, this );

    *result = TBDDRET_DEFAULT;    //drop-down was handled, also
    // TBDDRET_NODEFAULT drop-down was not handled.
    // TBDDRET_TREATPRESSED treat click as a button press
}
```

5.7 实例13：在状态栏中添加图标

1. 目标

如图5-7所示在状态栏上添加一个图标。

2. 策略

正如菜单可以自绘制一样，状态栏上的窗格也可以做到这一点。因此本例将为状态栏增加新的窗格并改变其风格为自绘制类型。在本实例中，将根据状态标记，使用两个LED图标之一来绘制该窗格。

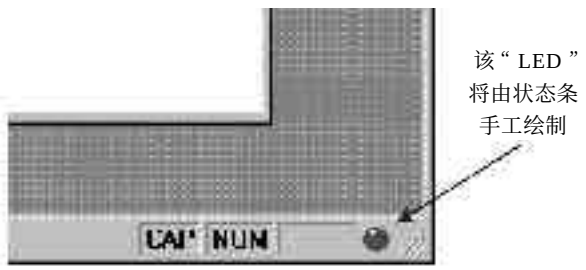

图5-7 状态栏上的图标

3. 步骤

1) 创建新资源

用Icon Editor创建两个或者多个图标以指示某些应用程序的不同状态。在本实例中将创建一个红色和一个绿色的 LED图标。

用String Table Editor创建新的空白字符串。在该字符串中留多少空白将确定状态窗格的宽度。本实例中用ID_INDICATOR_RED来标识该字符串。

2) 创建新的状态栏类

用ClassWizard创建新的 CStatusBarCtrl派生类，然后用 Text Editor将全部引用从CStatusBarCtrl更改为CStatusBar (ClassWizard不支持CStatusBar)。

在该新类中嵌入HICON变量并在构造函数中装载新图标：

```
CWzdStatusBar::CWzdStatusBar()
{
   // load any graphics
   m_hRedLedIcon = AfxGetApp() -> LoadIcon( IDI_RED_LED );
   m_hGreenLedIcon = AfxGetApp() -> LoadIcon( IDI_GREEN_LED );
}
```

这一步需要为欲绘制的窗格应用自绘制风格。将该功能封装到类函数 InitDrawing()中：

```
void CWzdStatusBar::InitDrawing()
{
   UINT nID, nStyle;
   int nPane, nWidth;

   // for each pane that we will be drawing
   nPane = CommandToIndex( ID_INDICATOR_LED );
   GetPaneInfo( nPane, nID, nStyle, nWidth );
   SetPaneInfo( nPane, nID, SBPS_OWNERDRAW|SBPS_NOBORDERS, nWidth );
}
```

这一步重载该类的DrawItem()函数来绘制该窗格。通过从DRAWITEMSTRUCT调用参数得到设备环境、窗格ID和要绘制的窗格区域等信息来启动该功能:

```
void CWzdStatusBar::DrawItem( LPDRAWITEMSTRUCT lpDrawItemStruct )
{
   // get graphic context to draw to
   CDC* pDC = CDC::FromHandle( lpDrawItemStruct -> hDC) ;

   // get the pane's rectangle
   CRect rect( lpDrawItemStruct -> rcItem );

   // get the pane's id
   UINT nID, nStyle;
   int nWidth;
   GetPaneInfo( lpDrawItemStruct -> itemID, nID, nStyle, nWidth );
```

由于该类中需要绘制一个以上的窗格，因此为窗格 ID创建一个开关变量并根据某种条件绘制两种图标之一。注意首先用按钮表面的当前颜色填充背景区域以便所绘制的内容与状态栏的剩余部分相协调:

```
   // draw to that pane based on a status
   switch ( nID )
   {
      case ID_INDICATOR_LED:
      // draw the background
      CBrush bkColor( GetSysColor( COLOR_3DFACE ) );
      CBrush* pOldBrush = ( CBrush* )pDC -> SelectObject( &bkColor );
      pDC -> FillRect( rect, &bkColor );
      pDC -> SelectObject( pOldBrush );

      // draw the appropriate LED
      HICON hicon = ( ( CMainFrame* )AfxGetMainWnd() ) -> m_bTest?
```

```
            m_hRedLedIcon:m_hGreenLedIcon;
            pDC -> DrawIcon( rect.left,rect.top,hicon );
            break;
        }
    }   // end of DrawItem()
```

参考本实例结尾的清单 —— 状态栏类可以查看状态栏类的完整代码列表。

3) 实现新的状态栏类

用该新类的名字替代Mainfrm.h中的CStatusBar。

在Mainfrm.cpp表中想让该窗格出现的位置增加新的状态窗格 ID：

```
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
    ID_INDICATOR_LED,        <<<<<< ADD HERE
};
```

在CMainFrame的OnCreate()函数中调用InitDrawing()函数初始化新的状态栏类:

```
// change selected pane style(s) to self-draw
m_wndStatusBar.InitDrawing();
```

现在可以打开状态栏。在想改变LED的状态时，应该使状态栏无效以使其重新绘制：

```
// change status
m_bTest = FALSE;

// force icon to be re-drawn
m_wndStatusBar.Invalidate();
```

4. 注意

可以改变图标的大小以使其适合状态栏。确保使用 32×32图标大小，但仅能用于左上角。这里当然还可以使用位图，但是图标比较适合这种方案，因为它一般较小并自动具有透明颜色，这将允许图标融合到状态栏中，并产生图标仿佛是状态栏一部分的视觉效果。

5. 使用光盘时注意

执行随书附带光盘上的工程时，可以注意到状态栏中具有绿色或者红色的图标。单击Test/Wzd菜单命令可以改变其颜色。

6. 程序清单 —— 状态栏类

```
#if !defined( AFX_WZDSTATB_H__5DF01360_876F_11D2_A18D_D6622706D73F__INCLUDED_ )
#define AFX_WZDSTATB_H__5DF01360_876F_11D2_A18D_D6622706D73F__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CWzdStatusBar window
```

```
class CWzdStatusBar : public CStatusBar
{
// Construction
public:
    CWzdStatusBar();

// Attributes
public:

// Operations
public:

    void InitDrawing();

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdStatusBar )
    // }}AFX_VIRTUAL
    virtual void DrawItem( LPDRAWITEMSTRUCT lpDrawItemStruct );

// Implementation
public:
    virtual ~CWzdStatusBar();

    // Generated message map functions
protected:
    // {{AFX_MSG( CWzdStatusBar )
    // NOTE - the ClassWizard will add and remove member functions here.
    // }}AFX_MSG

    DECLARE_MESSAGE_MAP()
private:
    HICON    m_hRedLedIcon;
    HICON    m_hGreenLedIcon;
};

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

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

#endif
    // !defined( AFX_WZDSTATB_H__5DF01360_876F_11D2_A18D_D6622706D73F__INCLUDED_ )
// WzdStatB.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdStatB.h"
#include "MainFrm.h"
```

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

//////////////////////////////////////////////////////////////////////////
// CWzdStatusBar

CWzdStatusBar::CWzdStatusBar()
{
    // load any graphics
    m_hRedLedIcon = AfxGetApp() -> LoadIcon( IDI_RED_LED );
    m_hGreenLedIcon = AfxGetApp() -> LoadIcon( IDI_GREEN_LED );

}

CWzdStatusBar::~CWzdStatusBar()
{
}


BEGIN_MESSAGE_MAP( CWzdStatusBar, CStatusBar )
    // {{AFX_MSG_MAP( CWzdStatusBar )
    // NOTE - the ClassWizard will add and remove mapping macros here.
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
// CWzdStatusBar message handlers

void CWzdStatusBar::InitDrawing()
{
    UINT nID,nStyle;
    int nPane,nWidth;

    // for each pane that we will be drawing
    nPane = CommandToIndex( ID_INDICATOR_LED );
    GetPaneInfo( nPane, nID, nStyle, nWidth );
    SetPaneInfo( nPane, nID, SBPS_OWNERDRAW|SBPS_NOBORDERS, nWidth );

}

void CWzdStatusBar::DrawItem( LPDRAWITEMSTRUCT lpDrawItemStruct )
{
    // get graphic context to draw to
    CDC* pDC = CDC::FromHandle( lpDrawItemStruct -> hDC );

    // get the pane's rectangle
    CRect rect( lpDrawItemStruct -> rcItem );
```

```
    // get the pane's id
    UINT nID,nStyle;
    int nWidth;

    GetPaneInfo( lpDrawItemStruct -> itemID, nID, nStyle, nWidth );

    // draw to that pane based on a status
    switch ( nID )
    {
       case ID_INDICATOR_LED:
          // draw the background
          CBrush bkColor( GetSysColor( COLOR_3DFACE ) );
          CBrush* pOldBrush = ( CBrush* )pDC -> SelectObject( &bkColor );
          pDC -> FillRect( rect, &bkColor );
          pDC -> SelectObject( pOldBrush );

          // draw the appropriate LED
          HICON hicon = ( ( CMainFrame* )AfxGetMainWnd() ) -> m_bTest?
             m_hRedLedIcon:m_hGreenLedIcon;
          pDC -> DrawIcon( rect.left,rect.top,hicon );
          break;
    }
}
```

5.8 实例14：使用伸缩条

1. 目标

如图5-8所示为应用程序增加伸缩条。

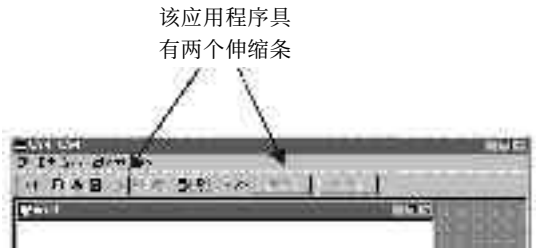

图5-8 伸缩条

2. 策略

MFC的新CReBar和CReBarCtrl类可以自动完成为应用程序添加伸缩条的大多数工作。但是这些类只能在 6.0版本中使用，而且伸缩条仅能用于Windows98系统或者安装了IE4.01及其以上版本的系统。

3. 步骤

1) 创建伸缩条

在CMainFrame类中嵌入伸缩条类：

```
CReBar  m_wndReBar;
```

在CMainFrame的OnCreate()函数中创建伸缩条的窗口：

```
if ( !m_wndReBar.Create( this ) )
{
   TRACE0( "Failed to create rebar\n" );
   return -1;            // fail to create
}
```

2) 为伸缩条增加控件

用如下AddBar()函数为该伸缩条增加工具栏：

```
m_wndReBar.AddBar( &m_wndToolBar,
    "WZD",                                  // rebar title
    NULL,                                   // a CBitmap background
    RBBS_GRIPPERALWAYS | RBBS_FIXEDBMP      // style
);
```

不要为该工具栏启动停靠功能或使用 ASSERT。工具栏不需要停靠 —— 伸缩条具有允许用户在其内部移动控件条的自动功能。

为了在伸缩条中添加其他类型的控件，将其单独或者连同其他一个或者多个控件放入对话框模板，然后为该模板创建对话条类并在 CMainFrame中嵌入对话条类变量，接着创建对话条并将其添加到伸缩条内：

```
if ( !m_WzdDialogBar.Create( this, IDD_WZD_DIALOG,
    WS_CHILD|WS_VISIBLE,-1 ) || !m_WzdDialogBar.InitDialog() )
{
    TRACE0( "Failed to create dialog bar\n" );
    return -1;              // fail to create
}

m_wndReBar.AddBar( &m_WzdDialogBar,
    “WZD”,                                  // rebar title
    NULL,                                   // a CBitmap background
    RBBS_GRIPPERALWAYS | RBBS_FIXEDBMP      // style
);
```

4. 注意

伸缩条的最大优点是允许用户定制自己的工具栏和其他控件的位置。 MFC应用程序已经具有使用停靠栏的功能。因而伸缩条也许是微软为非 MFC应用程序提供的功能，这也就是为什么不能使工具栏既有停靠功能同时又包括在伸缩条中的原因。作为 MFC应用程序的开发者，实际上为使工具栏停靠可以避免使用伸缩条，除非为了达到模仿特定外观的目的。

5. 使用光盘时注意

执行随书附带光盘上的工程时，可以注意到应用程序顶部的伸缩条内有一个工具栏和对话条，这些控件条可以通过拖动它们前面的提手条而移动。