```
    pDC -> DPtoLP (&org);

    CDC dcMem;
    dcMem.CreateCompatibleDC (pDC);
    CBitmap * pOldBitmap = dcMem.SelectObject (this);
    dcMem.SetMapMode (pDC -> GetMapMode ());

    pDC -> BitBlt (x, y, size.x, size.y, &dcMem, org.x, org.y, SRCCOPY);

    dcMem.SelectObject (pOldBitmap);
}

void CMaskedBitmap::DrawTransparent(CDC * pDC, int x, int y,
    COLORREF clrTransparency)
{
    BITMAP bm;
    GetBitmap (&bm);
    CPoint size (bm.bmWidth, bm.bmHeight);
    pDC -> DPtoLP (&size);

    CPoint org (0, 0);
    pDC -> DPtoLP (&org);

    //
    // Create a memory DC (dcImage) and select the bitmap into it.
    //
    CDC dcImage;
    dcImage.CreateCompatibleDC (pDC);
    CBitmap * pOldBitmapImage = dcImage.SelectObject (this);
    dcImage.SetMapMode (pDC -> GetMapMode ());

    //
    // Create a second memory DC (dcAnd) and in it create an AND mask.
    //
    CDC dcAnd;
    dcAnd.CreateCompatibleDC (pDC);
    dcAnd.SetMapMode (pDC -> GetMapMode ());

    CBitmap bitmapAnd;
    bitmapAnd.CreateBitmap (bm.bmWidth, bm.bmHeight, 1, 1, NULL);
    CBitmap * pOldBitmapAnd = dcAnd.SelectObject (&bitmapAnd);

    dcImage.SetBkColor (clrTransparency);
    dcAnd.BitBlt (org.x, org.y, size.x, size.y, &dcImage, org.x, org.y,
        SRCCOPY);
```

```
//
// Create a third memory DC (dcXor) and in it create an XOR mask.
//
CDC dcXor;
dcXor.CreateCompatibleDC (pDC);
dcXor.SetMapMode (pDC -> GetMapMode ());

CBitmap bitmapXor;
bitmapXor.CreateCompatibleBitmap (&dcImage, bm.bmWidth, bm.bmHeight);
CBitmap * pOldBitmapXor = dcXor.SelectObject (&bitmapXor);

dcXor.BitBlt (org.x, org.y, size.x, size.y, &dcImage, org.x, org.y,
    SRCCOPY);

dcXor.BitBlt (org.x, org.y, size.x, size.y, &dcAnd, org.x, org.y,
    0x220326);

//
// Copy the pixels in the destination rectangle to a temporary
// memory DC (dcTemp).
//
CDC dcTemp;
dcTemp.CreateCompatibleDC (pDC);
dcTemp.SetMapMode (pDC -> GetMapMode ());

CBitmap bitmapTemp;
bitmapTemp.CreateCompatibleBitmap (&dcImage, bm.bmWidth, bm.bmHeight);
CBitmap * pOldBitmapTemp = dcTemp.SelectObject (&bitmapTemp);

dcTemp.BitBlt (org.x, org.y, size.x, size.y, pDC, x, y, SRCCOPY);

//
// Generate the final image by applying the AND and XOR masks to
// the image in the temporary memory DC.
//
dcTemp.BitBlt (org.x, org.y, size.x, size.y, &dcAnd, org.x, org.y,
    SRCAND);

dcTemp.BitBlt (org.x, org.y, size.x, size.y, &dcXor, org.x, org.y,
    SRCINVERT);

//
// Blit the resulting image to the screen.
//
pDC -> BitBlt (x, y, size.x, size.y, &dcTemp, org.x, org.y, SRCCOPY);
```

```
    //
    // Restore the default bitmaps.
    //
    dcTemp.SelectObject (pOldBitmapTemp);
    dcXor.SelectObject (pOldBitmapXor);
    dcAnd.SelectObject (pOldBitmapAnd);
    dcImage.SelectObject (pOldBitmapImage);
}
```

图 15-7　BitmapDemo 应用程序

Windows 98 和 Windows 2000 都支持一个新的 API 函数::TransparentBlt，其功能与 StretchBlt 等价，也接受透明颜色。与 BitmapDemo 的 DrawTransparent 函数相同，::TransparentBlt 也跳过了颜色与透明色相同的像素。我没有使用::TransparentBlt 是因为我希望 BitmapDemo 能与在 Windows 98 和 Windows 2000 系统上运行一样也可以在低级的系统上运行。选用这两个函数中的哪个取决于所开发软件的目标平台。

15.2.7　编写 BMP 文件查看器

由 BitmapDemo 绘制的磁盘驱动器图像看上去效果很好，因为它是个简单的 16 位位图，其中颜色与系统调色板中的静态颜色匹配。只要是自己绘制的位图并始终选用默认调色板中的颜色，即使没有自定义 CPalettes，位图的显示效果也会不错。但是如果要编写一个读取由其他程序创建的任意 BMP 文件的应用程序，并且要依赖默认调色板对颜色的映射，那么包含 256 种或更多种颜色的位图就会得不到良好的显示，有的显示效果会更糟。通过创建 CPalette 使它的颜色与位图中的颜色相匹配，就可以在很大程度上改善图像的输出质量。本节中给出的示例程序就说明了这一点。该程序还介绍了一种方法，MFC 程序员可以将 CBitmap 和 DIB 扩展结合起来创建功能更强的位图。

图 15-8 所示的示例程序叫做 Vista。Vista 是一个文档/视图类型的 BMP 文件查看器，实际上它可以读取包含任意多种颜色的任何 BMP 文件，并在能显示 256 种或更多种颜色的屏幕上显示它们。(Vista 也可运行在 16 色屏幕上，但不要希望它能输出 16 种以外的颜色。)图 15-9 给出了其源程序的部分代码，这些程序非常简单。除了从磁盘上读取 BMP 文件并创建逻辑调色板的代码以外，应用程序几乎只包含作为文档/视图应用程序核心的标准程序代码。

视图的 OnDraw 函数通过以下方式将位图显示在屏幕上：将与位图关联的逻辑调色板选入设备描述表(如果这样的调色板存在的话)，并将位图位块传送给 CScrollView。OnDraw 通过调用文档的 GetPalette 函数来检索逻辑调色板，并调用文档的 GetBitmap 函数来检索位

图。GetPalette 返回一个指向调色板的 CPalette 指针,该调色板是在加载位图时由文档对象创建的。NULL 返回值说明没有调色板与位图关联,进而说明 Vista 正运行在非调色板化的视频适配器上。GetBitmap 返回一个指向位图的指针,该位图构成文档本身。Vista 的文档类 CVistaDoc 在名为 m_bitmap 的 CBitmap 数据成员中保存位图,在名为 m_palette 的 CPalette 数据成员中保存(如果有的话)伴随位图的调色板。在调用文档的 OnOpenDocument 函数时初始化位图和调色板对象(当用户从 File 菜单选中 Open 命令时),并在调用文档的 DeleteContents 函数时清除它们。

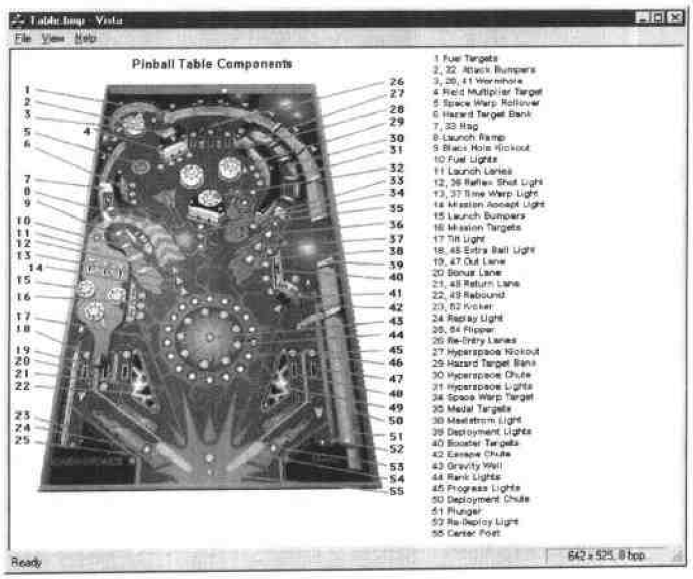

图 15-8 正在显示位图的 Vista 窗口

在 OnOpenDocument 中的一条简单的语句用于读取函数参数列表中指定的 BMP 文件:

```
HBITMAP hBitmap = (HBITMAP) ::LoadImage (NULL, lpszPathName,
    IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE|LR_CREATEDIBSECTION);
```

如果成功地创建了 DIB 扩展,由::LoadImage 返回的值就是有效的 HBITMAP,否则返回 NULL。如果::LoadImage 执行失败,很有可能是因为文件不包含 DIB。当::LoadImage 返回 NULL 时,OnOpenDocument 弹出消息框,指出出错原因。如果 HBITMAP 不是 NULL,OnOpenDocument 会将它附加给 m_bitmap。现在文档(位图)就得到了加载,几乎可以显示了。

如果 Vista 运行在调色板化的显示设备上,除非位图带有逻辑调色板,否则其输出效果可能不会太好。在::LoadImage 返回之后,OnOpenDocument 会捕获设备描述表并调用 GetDeviceCaps 来确定是否支持调色板。如果返回值不包含 RC_PALETTE 标志,OnOpenDocument 将立即返回,不对 m_palette 进行初始化。否则,OnOpenDocument 用逻辑调色板初始化 m_palette。

为确定创建调色板的最佳方式，OnOpenDocument 首先通过使用指向 DIBSECTION 结构的指针调用 GetObject 来了解位图包含有多少种颜色。DIBSECTION 结构有一个成员 BITMAPINFOHEADER 结构，BITMAPINFOHEADER 结构的 biClrUsed 和 biBitCount 字段显示了位图中包含的颜色种类。如果 biClrUsed 为非零值，它指出颜色种类。如果 biClrUsed 是 0，颜色种类就等于

```
1 << biBitCount
```

OnOpenDocument 的下列代码将 nColors 设置为位图中的颜色数：

```
DIBSECTION ds;
m_bitmap.GetObject (sizeof (DIBSECTION), &ds);
int nColors;
if (ds.dsBmih.biClrUsed != 0)
    nColors = ds.dsBmih.biClrUsed;
else
    nColors = 1 << ds.dsBmih.biBitCount;
```

接下来 OnOpenDocument 所要做的工作取决于 nColors 的值。如果 nColors 大于 256，说明位图具有的颜色深度为 16、24 或 32 位（保存在 BMP 文件中的图像总是使用 1 位、4 位、8 位、16 位、24 位或 32 位的颜色），OnOpenDocument 就会用以前获得的指向屏幕 DC 的指针来调用 CPalette::CreateHalftonePalette 创建一个半色调调色板：

```
if (nColors > 256)
    m_palette.CreateHalftonePalette (&dc);
```

接下来系统将用适合于设备描述表的彩虹色带创建一个一般的调色板。一般由 CreateHalftonePalette 创建的逻辑调色板都包含 256 种颜色。这可能不足以使包含成千上万种颜色的位图得以精确显示，但是与使用设备描述表的默认调色板相比，输出效果就好多了。

如果 nColors 小于或等于 256，OnOpenDocument 就会用颜色与位图中颜色匹配的逻辑调色板来初始化 m_palette。匹配位图颜色的关键是 API 函数::GetDIBColorTable，它将与 1 位、4 位或 8 位 DIB 扩展关联的颜色列表复制到一个 RGBQUAD 结构的数组中。然后用该数组初始化一个 PALETTEENTRY 结构的数组，并创建一个逻辑调色板：

```
RGBQUAD * pRGB = new RGBQUAD[nColors];

CDC memDC;
memDC.CreateCompatibleDC (&dc);
CBitmap * pOldBitmap = memDC.SelectObject (&m_bitmap);
::GetDIBColorTable ((HDC) memDC, 0, nColors, pRGB);
memDC.SelectObject (pOldBitmap);

UINT nSize = sizeof (LOGPALETTE) +
```

```
    (sizeof(PALETTEENTRY) * (nColors - 1));
LOGPALETTE* pLP = (LOGPALETTE*) new BYTE[nSize];

pLP->palVersion = 0x300;
pLP->palNumEntries = nColors;

for (int i = 0; i < nColors; i++) {
    pLP->palPalEntry[i].peRed = pRGB[i].rgbRed;
    pLP->palPalEntry[i].peGreen = pRGB[i].rgbGreen;
    pLP->palPalEntry[i].peBlue = pRGB[i].rgbBlue;
    pLP->palPalEntry[i].peFlags = 0;
}

m_palette.CreatePalette(pLP);
```

::GetDIBColorTable 只有在 DIB 扩展被选入设备之后才能使用,因此 OnOpenDocument 在调用它之前先创建了一个内存 DC 并将 m_bitmap 选入其中。剩下的就是一些细节工作了:为 LOGPALETTE 结构分配内存,将 RGBQUAD 值从颜色表传送到相应的 PALETTEENTRY 输入项,并调用 CreatePalette。一旦有了可用的调色板,Vista 就会以令人震惊的精确度在 256 色屏幕上显示大多数 256 色位图。

Vista 的状态栏中包含了一个信息读出项,标识了位图的尺寸和颜色深度(每个像素的位值)。在 OnOpenDocument 给 Vista 的主窗口发送一个 WM_USER_UPDATE_STATS 消息时状态栏得到更新,该消息包含一个指向需要在状态栏区域显示的字符串的指针。在框架窗口类中的消息处理程序响应消息并相应地对状态栏作出更新。

MainFrm.h

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

#if !defined(
    AFX_MAINFRM_H__3597FEA9_A70E_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_MAINFRM_H__3597FEA9_A70E_11D2_8E53_006008A82731__INCLUDED_

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

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)
```

```
// Attributes
public:

// Operations
public:

// 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 BOOL OnQueryNewPalette();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    //}}AFX_MSG
    afx_msg LRESULT OnUpdateImageStats(WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
};

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

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

#endif
// !defined(AFX_MAINFRM_H__3597FEA9_A70E_11D2_8E53_006008A82731__INCLUDED_)
```

MainFrm.cpp

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

```
#include "stdafx.h"
#include "Vista.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] =    FILE   ;
#endif
////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_WM_QUERYNEWPALETTE()
    ON_WM_PALETTECHANGED()
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_USER_UPDATE_STATS, OnUpdateImageStats)
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,
    ID_SEPARATOR
};

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

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

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

    //
    // Create the status bar.
    //
```

```
    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
    }
    //
    // Size the status bar's rightmost pane to hold a text string.
    //
    TEXTMETRIC tm;
    CClientDC dc(this);
    CFont* pFont = m_wndStatusBar.GetFont();
    CFont* pOldFont = dc.SelectObject(pFont);
    dc.GetTextMetrics(&tm);
    dc.SelectObject(pOldFont);

    int cxWidth;
    UINT nID, nStyle;
    m_wndStatusBar.GetPaneInfo(1, nID, nStyle, cxWidth);
    cxWidth = tm.tmAveCharWidth * 24;
    m_wndStatusBar.SetPaneInfo(1, nID, nStyle, cxWidth);
    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    return TRUE;
}

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

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

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

#endif //_DEBUG
```

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

BOOL CMainFrame::OnQueryNewPalette()
{
    CDocument * pDoc = GetActiveDocument ();
    if (pDoc != NULL)
        GetActiveDocument () -> UpdateAllViews (NULL);
    return TRUE;
}

void CMainFrame::OnPaletteChanged(CWnd * pFocusWnd)
{
    if (pFocusWnd != this) {
        CDocument * pDoc = GetActiveDocument ();
        if (pDoc != NULL)
            GetActiveDocument () -> UpdateAllViews (NULL);
    }
}

LRESULT CMainFrame::OnUpdateImageStats (WPARAM wParam, LPARAM lParam)
{
    m_wndStatusBar.SetPaneText (1, (LPCTSTR) lParam, TRUE);
    return 0;
}
```

VistaDoc.h

```
// VistaDoc.h : interface of the CVistaDoc class
//
////////////////////////////////////////////////////////////////////////////

#if !defined(
    AFX_VISTADOC_H__3597FEAB_A70E_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_VISTADOC_H__3597FEAB_A70E_11D2_8E53_006008A82731__INCLUDED_

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

class CVistaDoc : public CDocument
{
protected: // create from serialization only
    CVistaDoc();
    DECLARE_DYNCREATE(CVistaDoc)

// Attributes
```

```
public:
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CVistaDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
    virtual void DeleteContents();
    //}}AFX_VIRTUAL

// Implementation
public:
    CPalette * GetPalette();
    CBitmap * GetBitmap();
    virtual ~CVistaDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    CPalette m_palette;
    CBitmap m_bitmap;
    //{{AFX_MSG(CVistaDoc)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

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

#endif
// !defined(
//    AFX_VISTADOC_H__3597FEAB_A70E_11D2_8E53_006008A82731__INCLUDED_)
```

VistaDoc.cpp

```
// VistaDoc.cpp : implementation of the CVistaDoc class
//

#include "stdafx.h"
#include "Vista.h"

#include "VistaDoc.h"

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

/////////////////////////////////////////////////////////////////////////////
// CVistaDoc

IMPLEMENT_DYNCREATE(CVistaDoc, CDocument)

BEGIN_MESSAGE_MAP(CVistaDoc, CDocument)
    //{{AFX_MSG_MAP(CVistaDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVistaDoc construction/destruction

CVistaDoc::CVistaDoc()
{
}

CVistaDoc::~CVistaDoc()
{
}

BOOL CVistaDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    return TRUE;

/////////////////////////////////////////////////////////////////////////////
// CVistaDoc serialization
```

```
void CVistaDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////////////////
// CVistaDoc diagnostics

#ifdef _DEBUG
void CVistaDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CVistaDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CVistaDoc commands

BOOL CVistaDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    if (!CDocument::OnOpenDocument(lpszPathName))
        return FALSE;

    //
    // Open the file and create a DIB section from its contents.
    //
    HBITMAP hBitmap = (HBITMAP)::LoadImage(NULL, lpszPathName,
        IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE|LR_CREATEDIBSECTION);

    if (hBitmap == NULL)
    {
        CString string;
        string.Format(_T("%s does not contain a DIB"), lpszPathName);
        AfxMessageBox(string);
        return FALSE;
    }
```

```
    m_bitmap.Attach (hBitmap);

    //
    // Return now if this device doesn't support palettes.
    //
    CClientDC dc (NULL);
    if ((dc.GetDeviceCaps (RASTERCAPS) & RC_PALETTE) == 0)
        return TRUE;

    //
    // Create a palette to go with the DIB 扩展.
    //
    if ((HBITMAP) m_bitmap != NULL) {
        DIBSECTION ds;
        m_bitmap.GetObject (sizeof (DIBSECTION), &ds);

        int nColors;
        if (ds.dsBmih.biClrUsed != 0)
            nColors = ds.dsBmih.biClrUsed;
        else
            nColors = 1 << ds.dsBmih.biBitCount;

        //
        // Create a halftone palette if the DIB 扩展 contains more
        // than 256 colors.
        //
        if (nColors > 256)
            m_palette.CreateHalftonePalette (&dc);

        //
        // Create a custom palette from the DIB 扩展's color table
        // if the number of colors is 256 or less.
        //
        else {
            RGBQUAD* pRGB = new RGBQUAD[nColors];

            CDC memDC;
            memDC.CreateCompatibleDC (&dc);
            CBitmap* pOldBitmap = memDC.SelectObject (&m_bitmap);
            ::GetDIBColorTable ((HDC) memDC, 0, nColors, pRGB);
            memDC.SelectObject (pOldBitmap);
            UINT nSize = sizeof (LOGPALETTE) +
                (sizeof (PALETTEENTRY) * (nColors - 1));
            LOGPALETTE* pLP = (LOGPALETTE*) new BYTE[nSize];

            pLP->palVersion = 0x300;
            pLP->palNumEntries = nColors;

            for (int i = 0; i < nColors; i++) {
```

```
                pLP->palPalEntry[i].peRed = pRGB[i].rgbRed;
                pLP->palPalEntry[i].peGreen = pRGB[i].rgbGreen;
                pLP->palPalEntry[i].peBlue = pRGB[i].rgbBlue;
                pLP->palPalEntry[i].peFlags = 0;
            }

            m_palette.CreatePalette(pLP);
            delete[] pLP;
            delete[] pRGB;
        }
    }
    return TRUE;
}

void CVistaDoc::DeleteContents()
{
    if ((HBITMAP) m_bitmap != NULL)
        m_bitmap.DeleteObject();

    if ((HPALETTE) m_palette != NULL)
        m_palette.DeleteObject();

    CDocument::DeleteContents();
}

CBitmap* CVistaDoc::GetBitmap()
{
    return ((HBITMAP) m_bitmap == NULL) ? NULL : &m_bitmap;
}

CPalette* CVistaDoc::GetPalette()
{
    return ((HPALETTE) m_palette == NULL) ? NULL : &m_palette;
}
```

VistaView.h

```
// VistaView.h : interface of the CVistaView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(
    AFX_VISTAVIEW_H__359A7FEAD__A70E__11D2__8E53__006008A82731__INCLUDED_)
#define AFX_VISTAVIEW_H__359A7FEAD_A70E_11D2_8E53_006008A82731__INCLUDED_

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

```
class CVistaView : public CScrollView
{
protected: // create from serialization only
    CVistaView();
    DECLARE_DYNCREATE(CVistaView)

// Attributes
public:
    CVistaDoc * GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CVistaView)
    public:
    virtual void OnDraw(CDC * pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual void OnInitialUpdate(); // called first time after construct
    //}}AFX_VIRTUAL

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

// Generated message map functions
protected:
    //{{AFX_MSG(CVistaView)
        // NOTE-the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in VistaView.cpp
inline CVistaDoc * CVistaView::GetDocument()
   { return (CVistaDoc * )m_pDocument; }
#endif

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

//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif
// !defined(
//     AFX_VISTAVIEW_H__3597FEAD_A70E_11D2_8E53_006008A82731__INCLUDED_)
```

VistaView.cpp

```
// VistaView.cpp : implementation of the CVistaView class
//

#include "stdafx.h"
#include "Vista.h"

#include "VistaDoc.h"
#include "VistaView.h"

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

/////////////////////////////////////////////////////////////////////////////
// CVistaView

IMPLEMENT_DYNCREATE(CVistaView, CScrollView)

BEGIN_MESSAGE_MAP(CVistaView, CScrollView)
    //{{AFX_MSG_MAP(CVistaView)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVistaView construction/destruction

CVistaView::CVistaView()
{
}

CVistaView::~CVistaView()
{
}

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

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

////////////////////////////////////////////////////////////////////////////
// CVistaView drawing

void CVistaView::OnDraw(CDC * pDC)
{
    CVistaDoc * pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CBitmap * pBitmap = pDoc -> GetBitmap ();

    if (pBitmap != NULL) {
        CPalette * pOldPalette;
        CPalette * pPalette = pDoc -> GetPalette ();

        if (pPalette != NULL) {
            pOldPalette = pDC -> SelectPalette (pPalette, FALSE);
            pDC -> RealizePalette ();
        }
        DIBSECTION ds;
        pBitmap -> GetObject (sizeof (DIBSECTION), &ds);

        CDC memDC;
        memDC.CreateCompatibleDC (pDC);
        CBitmap * pOldBitmap = memDC.SelectObject (pBitmap);

        pDC -> BitBlt (0, 0, ds.dsBm.bmWidth, ds.dsBm.bmHeight, &memDC,
            0, 0, SRCCOPY);

        memDC.SelectObject (pOldBitmap);

        if (pPalette != NULL)
            pDC -> SelectPalette (pOldPalette, FALSE);
    }
}

void CVistaView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate ();

    CString string;
    CSize sizeTotal;
    CBitmap * pBitmap = GetDocument () -> GetBitmap ();

    //
    // If a bitmap is loaded, set the view size equal to the bitmap size.
    // Otherwise, set the view's width and height to 0.
    //
```

```
    if (pBitmap != NULL) {
        DIBSECTION ds;
        pBitmap->GetObject (sizeof (DIBSECTION), &ds);
        sizeTotal.cx = ds.dsBm.bmWidth;
        sizeTotal.cy = ds.dsBm.bmHeight;
        string.Format (_T ("\t%d x %d, %d bpp"), ds.dsBm.bmWidth,
            ds.dsBm.bmHeight, ds.dsBmih.biBitCount);
    }
    else {
        sizeTotal.cx = sizeTotal.cy = 0;
        string.Empty ();
    }

    AfxGetMainWnd ()->SendMessage (WM_USER_UPDATE_STATS, 0,
        (LPARAM) (LPCTSTR) string);
    SetScrollSizes (MM_TEXT, sizeTotal);
}

/////////////////////////////////////////////////////////////////////////////
// CVistaView diagnostics

#ifdef _DEBUG
void CVistaView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CVistaView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

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

/////////////////////////////////////////////////////////////////////////////
// CVistaView message handlers
```

图 15-9　Vista 应用程序

15.2.8 再论::LoadImage

Vista 用如此少的代码完成如此多的工作,其内在原因是::LoadImage 函数用一条语句就可以从 BMP 文件创建一个 DIB 扩展,该语句为:

```
HBITMAP hBitmap = (HBITMAP) ::LoadImage (NULL, lpszPathName,
IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE|LR_CREATEDIBSECTION);
```

::LoadImage 对于 DIB 扩展就相当于::LoadBitmap 和 CDC::LoadBitmap 对于 DDB。但还不仅是这些。在此对其输入参数不作一一介绍,读者可以从有关文献中了解这些内容,下面要对::LoadImage 的用途作一个简要的总结:

- 加载位图资源,并由此创建 DDB 和 DIB 扩展。
- 加载保存在 BMP 文件中的位图,并由此创建 DDB 和 DIB 扩展。
- 在图像加载时自动将 3 种灰色调(RGB (128,128,128)、RGB (192,192,192)以及 RGB (223,223,223))转换为系统颜色 COLOR_3DSHADOW、COLOR_3DFACE 和 COLOR_3DLIGHT。
- 自动将位图左上角像素的颜色转换为系统颜色 COLOR_WINDOW 或 COLOR_3DFACE,使得与之相近的像素在 COLOR_WINDOW 或 COLOR_3DFACE 背景上不可见。
- 将彩色图像转换为单色图像。

要记住::LoadImage 的颜色映射能力只适用于包含 256 种或更少种类颜色的位图。带有 256 种或更少种类颜色的 DIB 包含内置的颜色表,使得颜色映射可以快速而高效地执行。::LoadImage 只是简单地修改颜色表,而不是通过检查图像中的每个像素来进行颜色转换的。

Vista 说明了用::LoadImage 从 BMP 文件创建一个 DIB 扩展并将其附加给 CBitmap 对象的方法。将位图作为 DIB 扩展而不是普通的 DDB 来加载的好处,在于可以对它调用如::GetDIBColorTable 这样的函数。如果在调用::LoadImageon 时忽略了 LR_CREATEDIB-SECTION 标志,就不能访问位图的颜色表,并以此创建逻辑调色板了。一般而言,如果从现在开始您尽可能地使用 DIB 扩展而不是 DDB,您的应用程序就会更容易地移植到将来版本的 Windows 中(并且可能执行效果也会更好)。

15.3 区域

MFC 的 CRect 类代表简单的矩形区域,这是用相互垂直的四条边围成的空间。更复杂的空间区域可以用 CRgn 类表示,它封装了通常大致称为"区域"的 GDI 对象。区域的基本用途是用来创建复杂图案,用作 CDI 画图函数中的剪切边框。当然还可以以其他方式使用

CRgn。下面简要介绍一下区域和区域的用途。

15.3.1　区域和 CRgn 类

CRgn 提供了一些函数可以用来生成几何形状的区域，将已有的区域组合来创建更复杂的区域，以及执行其他别的操作，如对区域进行命中测试或检索区域的边界矩形。一旦创建区域，CDC 类就可以用区域提供的工具使用区域画图了，例如：用一种画刷颜色填充区域，或用它来剪切其他画图操作。这里首先我们将讲述区域的创建过程，然后看一下那些用到区域的 CDC 函数，最后通过开发一个示例程序来结束本节内容，该程序将使用区域来生成一些非同寻常的输出效果。

创建区域

在构造了 CRgn 对象之后，通过调用 CRgn 类为创建区域提供的几个成员函数之一就可以创建区域，同时创建的区域也归属于该对象。在表 15-4 中总结了一些相关的 CRgn 函数。

表 15-4　CRgn 创建区域函数

函　数	说　明
CreateRectRgn	由一组坐标值创建一个矩形区域
CreateRectRgnIndirect	由 RECT 结构或 CRect 对象创建一个矩形区域
CreateEllipticRgn	由一组坐标值创建一个椭圆形区域
CreateEllipticRgnIndirect	由 RECT 结构或 CRect 对象创建一个椭圆形区域
CreateRoundRectRgn	创建一个带有圆角的矩形区域
CreatePolygonRgn	由一组坐标值创建一个多边形区域
CreatePolyPolygonRgn	从一组坐标值创建由多个多边形组成的区域
CreateFromPath	根据通路创建一个区域
CreateFromData	通过给已存在的区域进行二维坐标变化来创建一个区域
CopyRgn	创建一个区域，它是已存在区域的拷贝

大多数这些函数都可以直接使用。例如：要使用名为 rect 的 CRect 对象创建一个椭圆形区域，其中 rect 定义了椭圆区域的边框，可用如下语句实现：

```
CRgn rgn;
rgn.CreateEllipticRgnIndirect(&rect);
```

要创建一个带有圆角的矩形区域，可以用以下方式实现：

```
CRgn rgn;
rgn.CreateRoundRectRgn(rect.left, rect.top, rect.right,
    rect.bottom, nCornerWidth, nCornerHeight);
```

nCornerWidth 和 nCornerHeight 分别代表用来使角圆滑的椭圆的水平和垂直尺寸。所有传给

创建区域函数的坐标值都是逻辑坐标。和其他 GDI 对象一样,在区域不再需要时必须删除。如果在堆栈上创建 CRgn,删除操作会自动执行,因为 CRgn 超出范围后它会删除与之链接的 GDI 区域。

功能最强大的区域创建函数之一是 CRgn::CreateFromPath,它可以将设备描述表的当前通路转换成区域。通路是通过在调用 CDC::BeginPath 和 CDC::EndPath 间调用其他 GDI 绘图函数生成的轮廓线。下列语句生成一个简单的椭圆通路,并将其转换为区域:

```
dc.BeginPath ();                    // Define a path.
dc.Ellipse (0, 0, 400, 200);
dc.EndPath ();

CRgn rgn;                   // Convert the path into a region.
rgn.CreateFromPath (&dc);
```

这段程序没什么特别的地方,我们通过简单地调用 CRgn::CreateEllipticRgn 就能完成相同的工作。但是 CreateFromPath 非凡的功能在于可以用非常复杂的对象创建通路,如使用贝塞尔曲线和文本轮廓线。下列语句根据文本字符串"Hello,MFC"中的字符创建了一个区域:

```
dc.BeginPath ();
dc.TextOut (0, 0, CString (_T ("Hello, MFC")));
dc.EndPath ();
```

一旦创建后,通路就可以用 CRgn::CreateFromPath 转换为区域了。Ellipse 和 TextOut 只是可以与 BeginPath 和 EndPath 共同使用的几个 CDC 绘图函数中的两个。有关 API 函数::BeginPath 在 MFC 文献中提供了一个可供参考的详细内容列表。(要注意,可以用来生成通路的 GDI 绘图函数的子集在 Windows 95 和 Windows 98 以及 Windows NT 和 Windows 2000 之间稍有不同。)还可以用与区域无关的方式来使用通路。要学习使用通路进行绘图操作,请参考有关 CDC 函数 FillPath、StrokePath、StrokeAndFillPath 和 WidenPath 的 MFC 文献。

创建复杂区域的另一种方法是用 CRgn::CombineRgn 组合已有的区域。CombineRgn 有三个参数:指向要组合的两个区域的 CRgn 指针(区域 1 和区域 2)和一个整数值以指定组合方式。组合方式可以是表 15-5 中列出的 5 种方式之一。

表 15-5 5 种组合方式

方式	说 明
RGN_COPY	设置区域等于区域 1
RGN_AND	设置区域等于区域 1 和区域 2 的交集
RGN_OR	设置区域等于区域 1 和 2 的并集
RGN_DIFF	设置区域等于区域 1 所围的面积减去区域 2 所围的面积
RGN_XOR	设置区域等于区域 1 和 2 的不重叠部分的面积

组合方式告诉 GDI 组合区域所使用的逻辑运算。语句

```
CRgn rgn1, rgn2, rgn3;
rgn1.CreateEllipticRgn (0, 0, 100, 100);
rgn2.CreateEllipticRgn (40, 40, 60, 60);
rgn3.CreateRectRgn (0, 0, 1, 1);
rgn3.CombineRgn (&rgn1, &rgn2, RGN_DIFF);
```

创建了一个由中间有孔的圆形成的环行区域。注意要对某区域调用 CombineRgn，就必须等到该区域用其他方法创建之后才能调用。（也就是说要等到 CRgn 具有它的 HRGN 之后）。这就是在本例中调用 CombineRgn 之前要调用 CreateRectRgn 创建一个小矩形区域的原因。

使用区域

在区域创建之后，可以用它来做什么呢？我们从下列使用区域的 CDC 绘图函数开始：

- CDC::FillRgn 用指定的刷子填充一个区域。
- CDC::PaintRgn 使用当前刷子填充区域。
- CDC::InvertRgn 反转区域中的颜色。
- CDC::FrameRgn 用指定的刷子给区域画边框。

还可以使用 CWnd::InvalidateRgn 使区域无效。在 Windows 内部就是使用区域而不是矩形来跟踪窗口的无效区域的。在调用 CDC::GetClipBox 时，得到的是包围窗口无效区域的矩形。这个区域可能只是一个简单的矩形，也可能是更复杂的形状。

在区域中使用 CRgn::PtInRegion 可以执行命中测试。假设在窗口客户区创建一个椭圆形区域。使用 PaintRgn 或 FillRgn 给区域填充了与窗口背景不同的颜色，现在希望知道何时用户在椭圆内部单击了鼠标左键。如果 m_rgn 是 CRgn 对象，则 OnLButtonDown 处理程序大概是如下样子：

```
void CMyWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    CClientDC dc (this);
    dc.DPtoLP (&point); // Convert to logical coordinates.
    if (m_rgn.PtInRegion (point)) {
        // The point falls within the region.
    }
}
```

MFC 的 CRect 类提供了一个功能相似的函数：PtInRect。事实上，在 API（以及 MFC 的成员函数中）中矩形和区域之间有许多相似的函数：InvalidateRect 和 InvalidateRgn、FillRect 和 FillRgn，等等。矩形函数执行起来速度较快，因此要尽量避免使用区域函数处理简单矩形，而应尽量使用等价的矩形函数。

对于复杂的图形图像，使用区域作为剪切边界有许多好处。使用 CDC::SelectObject 或

CDC::SelectClipRgn 可以把区域选入设备描述表。一旦选入,该区域就成为后来设备描述表上输出的剪切边界了。下一节给出的 RegionDemo 应用程序就是使用剪切区域来生成图像,如果用其他方法生成这种图像则实现起来非常困难。而若利用区域作为图形输出的虚拟模板,图像的实现就会非常简单。使用复杂剪切区域的缺点是其执行效率低。但是有时使用剪切区域是您得到期望输出的唯一方法。如果使用通路作为剪切区域,那就没必要将它先转换为区域然后再选入设备描述表。可以使用 CDC::SelectClipPath 直接将通路选入设备描述表。

区域的另一个更具有想象力的用途是:将其传递给 CWnd::SetWindowRgn 函数,使之成为窗口区域。对于整个窗口而言"窗口区域"是一个剪切区域。Windows 不允许窗口区域以外的元素对象被绘制,包括标题栏和其他非客户区窗口元素。创建一个椭圆区域并把它的句柄传递给 SetWindowRgn,就可以得到一个椭圆窗口。如果窗口是顶层窗口而且它的标题栏被视图隐藏了,则可以使用 OnNcHitTest 处理程序将 HTCLIENT 命中测试码转换为 HTCAPTION 码使得窗口可以通过客户区被拖动。非矩形窗口区域的更实际的用途是用来创建独具风格的文本冒泡窗口,事实上它确实是窗口,可以和其他窗口一样接收消息。在 SetWindowRgn 的帮助下,创建弹出窗口类并不非常困难,该类可以在形状像信息提示气球那样的窗口中显示帮助文本,当被单击时会自动关闭。

15.3.2 RegionDemo 应用程序

图 15-10 显示了应用程序 RegionDemo 的输出,该应用程序使用了剪切区域来绘制用射线线段排列形成的"Hello,MFC"。剪切区域是由通路生成的,通路又是通过在 CDC::BeginPath 和 CDC::EndPath 之间调用 CDC::TextOut 生成的。在 OnPaint 中完成了所有的工作。阅读一下图 15-11 中的源程序代码;对于输出每个阶段所执行的操作大部分都容易理解,不太容易理解的地方可能是在程序段中使用了两个不同的 CRgn 对象,并对 CRgn 成员函数执行了多种调用来生成最后的剪切区域(rgn1),用 CDC::SelectClipRgn 把该区域选入设备描述表。

图 15-10 RegionDemo 窗口

RegionDemo.h

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

class CMainWindow : public CFrameWnd
{
public:
    CMainWindow ();

protected:
    afx_msg void OnPaint ();
    DECLARE_MESSAGE_MAP ()
};
```

RegionDemo.cpp

```
#include <afxwin.h>
#include <math.h>
#include "RegionDemo.h"
CMyApp myApp;

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

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

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

BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()

CMainWindow::CMainWindow ()
{
    Create (NULL, _T ("Region Demo"));
```

```
}

void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);

    //
    // Create a 72-point Times New Roman font.
    //
    CFont font;
    font.CreatePointFont (720, _T ("Times New Roman"));

    //
    // Create a clipping region from the text string "Hello, MFC."
    //
    CRect rect;
    GetClientRect (&rect);
    CString string ("Hello, MFC");

    CFont * pOldFont = dc.SelectObject (&font);
    CSize size = dc.GetTextExtent (string);
    int x = (rect.Width () - size.cx) / 2;
    TEXTMETRIC tm;
    dc.GetTextMetrics (&tm);
    int y = (rect.Height () - tm.tmHeight) / 2;

    dc.BeginPath ();
    dc.TextOut (x, y, string);
    dc.EndPath ();
    dc.SelectObject (pOldFont);

    CRect rcText;
    CRgn rgn1, rgn2;
    rgn1.CreateFromPath (&dc);
    rgn1.GetRgnBox (&rcText);
    rgn2.CreateRectRgnIndirect (&rcText);
    rgn1.CombineRgn (&rgn2, &rgn1, RGN_DIFF);

    dc.SelectClipRgn (&rgn1);

    //
    // Draw a radial array of lines.
    //
    dc.SetViewportOrg (rect.Width () / 2, rect.Height () / 2);
    double fRadius = hypot (rect.Width () / 2, rect.Height () / 2);

    for (double fAngle = 0.0; fAngle < 6.283; fAngle += 0.01745) {
        dc.MoveTo (0, 0);
```

```
            dc.LineTo ((int) ((fRadius * cos (fAngle)) + 0.5),
                (int) ((fRadius * sin (fAngle)) + 0.5));
        }
    }
```

图 15-11　RegionDemo 应用程序

下面我们将一步一步地分析用来创建剪切区域的程序代码，剪切区域是在描绘文本字符串中字符轮廓的通路创建之后才被创建的。语句

```
rgn1.CreateFromPath (&dc);
```

用与通路匹配的区域初始化 rgn1。图 15-12 中显示了第一个区域的样子。区域内部指的是矩形中挖去字符"Hello,MFC"之后的部分。(一些图形系统，特别是 PostScript，对由字符轮廓形成的通路处理起来完全不同，使得区域的内部就是字符自身。GDI 的处理则完全相反，从字符的边框创建一个区域，然后将字符轮廓线包围的部分减去。)接下来，语句

```
rgn1.GetRgnBox (&rcText);
rgn2.CreateRectRgnIndirect (&rcText);
```

将 rgn1 的边框复制给名为 rcText 的 CRect 对象并创建一个区域(rgn2)。

最后一个语句实际上是通过从 rgn2 中减去 rgn1 而实现了对 rgn1 的反转：

```
rgn1.CombineRgn (&rgn2, &rgn1, RGN_DIFF);
```

得到的结果区域其内部准确地与用 TextOut 绘制的字符内部匹配。在将区域选入设备描述表之后，以 1°为增量从窗口客户区的中心向外绘制了射线。由于线段被区域边框剪切了，所以在字符轮廓以外不会绘制出射线。

图 15-12　由文本字符串"Hello,MFC"生成的通路

可以将生成区域的程序段从 OnPaint 中移到 OnCreate，这样可以使 RegionDemo 执行的效率稍微高一些。在每次重绘窗口时并不需要重新生成区域，但是为保证它位于窗口中间就需要使用 CRgn::OffsetRgn 重新安排它的位置。删除过多 CRgn 函数的调用在一定程度上可以提高输出速度，但是运行效率中最大的问题是在用区域的边界剪切屏幕上绘制的线段时产生的。在计算机绘图中使用复杂剪切区域的主要问题就出在运行效率上，因此除非再没有别的合理选择了，否则最好避免使用非矩形剪切区域。

第 16 章　公 用 控 件

自 1.0 版本之后，Microsoft Windows 提供了一套核心控件，包括按钮、单选按钮、列表框和其他公用用户界面(UI)对象。Windows 95 和 Windows NT 3.51 又扩展了可选控件集合，在 Comctl32.dll 中又增添了 15 种新控件类型。这些控件统称为“公用控件”，既包括简单的如进度控件，它能提供进度的图形反馈信息，又包括复杂的如树形视图控件，它分层展示树形

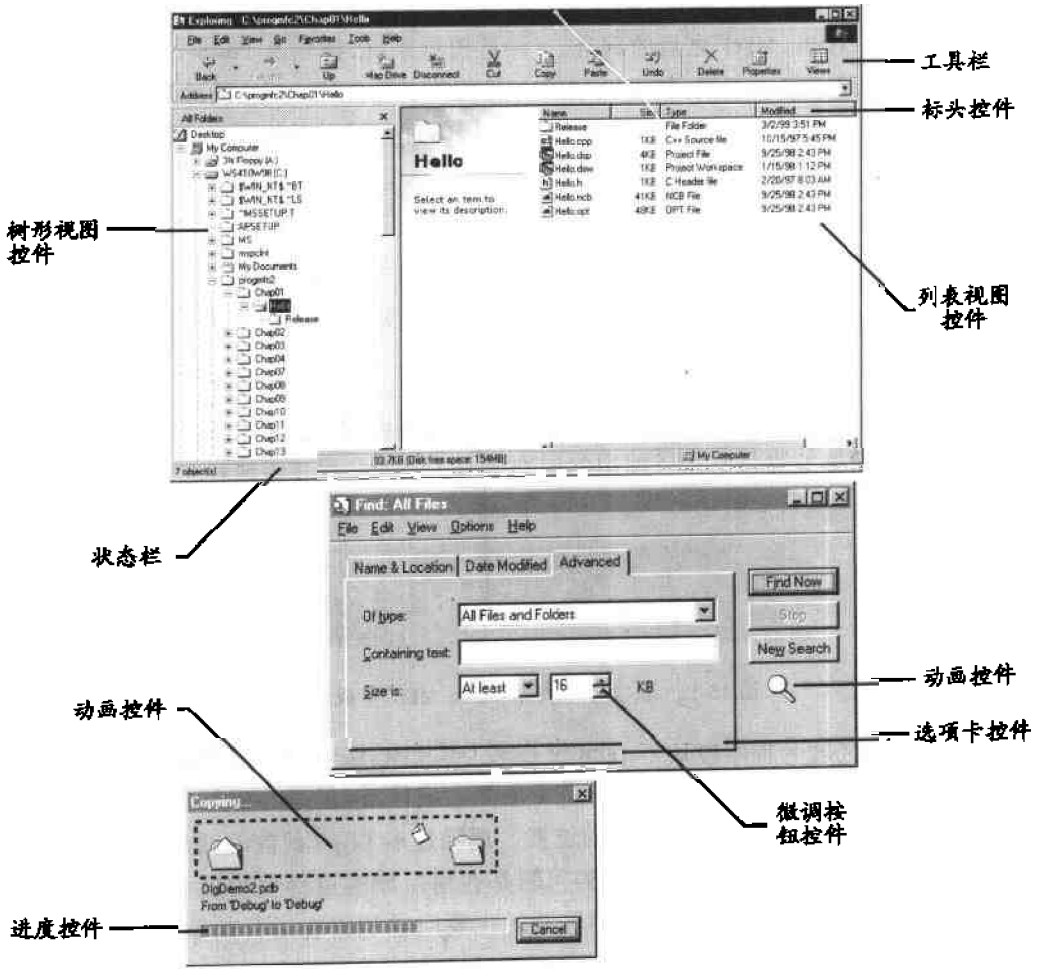

图 16-1　公用控件和 Windows 用户界面

结构的数据，该数据的分支能随鼠标单击展开和折叠。而 Microsoft Internet Explorer在 Comctl32.dll 中又增添了一些控件，它将当前平台支持的公用控件的总数增加到 20 个。Internet Explorer 提供的控件包括选择日期和时间的控件、输入 Internet Protocol (IP) 地址的控件，等等。

在 Windows 中到处都有公用控件。公用控件已经成为操作系统外观和性能的重要组成部分。图 16-1 展示了 Windows 使用部分公用控件的情况。资源管理器窗口中的标头控件是列表视图控件的一部分，但是也可以独立于列表视图创建标头控件。Find 执行搜索时，绕圈移动的放大镜是一个动画控件。同样，文件移动、拷贝或删除时，屏幕上飞过的纸张也是动画控件。后面您会了解到，动画控件通过播放以 Windows Audio Video Interleaved (AVI) 格式记录的序列，使简单的动画显示变得很容易。

本章将介绍公用控件和它们的 MFC 接口。首先我们整体介绍一下公用控件，然后介绍控件的创建方法和控件发送消息的独特方式。之后，我们详细介绍几个公用控件，并研究一下这些控件的示例程序。

16.1　公用控件基础

和提供类封装 User.exe 中实现的核心控件类型一样，MFC 也提供类封装公用控件。表 16-1 给出了 20 种公用控件，以及它们的 WNDCLASS 和相应的 MFC 类。

表 16-1　公 用 控 件

控件类型	WNDCLASS	WNDCLASS 别名	MFC 类
Animation	"SysAnimate32"	ANIMATE_CLASS	CAnimateCtrl
ComboBoxEx *	"ComboBoxEx32"	WC_COMBOBOXEX	CComboBoxEx
Date-Time *	"SysDateTimePick32"	DATETIMEPICK_CLASS	CDateTimeCtrl
Header	"SysHeader32"	WC_HEADER	CHeaderCtrl
Hotkey	"msctls_hotkey32"	HOTKEY_CLASS	CHotKeyCtrl
Image list	N/A	N/A	CImageList
IP address * *	"SysIPAddress32"	WC_IPADDRESS	CIPAddressCtrl
List view	"SysListView32"	WC_LISTVIEW	CListCtrl
Month calendar *	"SysMonthCal32"	MONTHCAL_CLASS	CMonthCalCtrl
Progress	"msctls_progress32"	PROGRESS_CLASS	CProgressCtrl
Property sheet	N/A	N/A	CPropertySheet
Rebar *	"ReBarWindow32"	REBARCLASSNAME	CReBarCtrl
Rich edit	"RichEdit20A"(ANSI) or "RichEdit20W" (Unicode)	RICHEDIT_CLASS	CRichEditCtrl
Slider	"msctls_trackbar32"	TRACKBAR_CLASS	CSliderCtrl

续表

控件类型	WNDCLASS	WNDCLASS 别名	MFC 类
Spin button	"msctls_updown32"	UPDOWN_CLASS	CSpinButtonCtrl
Status bar	"msctls_statusbar32"	STATUSCLASSNAME	CStatusBarCtrl
Tab	"SysTabControl32"	WC_TABCONTROL	CTabCtrl
Toolbar	"ToolbarWindow32"	TOOLBARCLASSNAME	CToolBarCtrl
ToolTip	"tooltips_class32"	TOOLTIPS_CLASS	CToolTipCtrl
Tree view	"SysTreeView32"	WC_TREEVIEW	CTreeCtrl

* 要求 Internet Explorer 3.0 或更高版本

* * 要求 Internet Explorer 4.0 或更高版本

该表还列出了在头文件 Commctrl.h 中给这些 WNDCLASS 定义的别名。可以想象,由于图像列表和属性页在严格意义上不是控件,所以它们没有 WNDCLASS,但由于它们的程序代码在 Comctl32.dll 里,故我们仍把它们看作公用控件。有时您会看到拖动列表框和公用控件列在一起。在这里我不想把它们混在一块,因为拖动列表框不是独立的控件;它们是传统列表框,只是被 Comctl32.dll 中的某个函数转换为"拖动"列表框了。MFC 在 CDragListBox 中提供了拖动列表框的一种简便实现方法,如感兴趣,请参见 CDragListBox 有关文档。

从表中可以得知,某些公用控件只能在安装了 Internet Explorer 特殊版本的系统上使用。这是因为在安装 Internet Explorer 时,安装程序自动更新 Comctl32.dll。在本章我会多次提到"该样式只能在配备了 Internet Explorer 3.0 或更高版本的系统中使用",或"该功能要求 Internet Explorer 4.0"这样的话。实际上,并不是需要 Internet Explorer 的支持,而是需要该版本 Internet Explorer携带的 Comctl32.dll 的支持。因为目前安装新版本 Internet Explorer 是得到最新版本 Comctl32.dll 的唯一合法方式,所以 Internet Explorer 是提供版本支持的基础。实际情况是:公用控件对已安装 Comctl32.dll 版本有多种依赖性,而一些系统根本没有安装 Internet Explorer,如果已知某功能需要某版本的 Comctl32.dll,您可能想了解怎样在运行时判断该功能是否得到支持。这里有一个简单例程,它可以返回 Comctl32.dll 的主、次版本号。如果主机系统中的 Comctl32.dll 版本比 Internet Explorer 3.0 早,则该例程返回 4.0;如果主机系统根本没有装 Comctl32.dll,则返回 0.0:

```
void GetComctlVersion(DWORD &dwMajor, DWORD &dwMinor)
{
  dwMajor = dwMinor = 0;
  HINSTANCE hLib = ::LoadLibrary (_T ("Comctl32.dll"));
  if (hLib != NULL) {
      DLLGETVERSIONPROC pDllGetVersion =
          (DLLGETVERSIONPROC) ::GetProcAddress (hLib,__T ("DllGetVersion"));
      if (pDllGetVersion) { // IE 3.0 or higher
          DLLVERSIONINFO dvi;
```

```
            ::ZeroMemory (&dvi, sizeof (dvi));
            dvi.cbSize = sizeof (dvi);
            HRESULT hr = (*pDllGetVersion) (&dvi);
            if (SUCCEEDED (hr)) {
                dwMajor = dvi.dwMajorVersion;
                dwMinor = dvi.dwMinorVersion;
            }
        }
        else { // Pre IE 3.0
            dwMajor = 4;
            dwMinor = 0;
        }
        ::FreeLibrary (hLib);
    }
}
```

这里，还需要一种方法将 Internet Explorer 版本号转换为 Comctl32.dll 版本号。表 16-2 会有所帮助。

表 16-2　Internet Explorer 与 Comctl32.dll 的对应版本号

Internet Explorer 版本	Comctl32.dll 版本
3.0	4.70
4.0	4.71
4.01	4.72

现在如果某功能需要 Internet Explorer 3.0 或更高版本的支持，而您又想知道运行时该功能是否得到支持，您可以这样做：

```
DWORD dwMajor, dwMinor;
GetComctlVersion (dwMajor, dwMinor);
if ((dwMajor == 4 && dwMinor >= 70)||dwMajor > 4) {
    // The feature is supported.
}
else {
    // The feature is not supported.
}
```

确实，这种方法缺陷很多。但它是当前唯一可用的方法。

16.1.1　创建公用控件

如果不用 API 函数，有两种方法可以用来创建公用控件。第一种方法是将相应的 MFC 控件类实例化，然后调用所生成对象的 Create 函数，如下所示：

```
#include <afxcmn.h>
```

```
.
.
.
CProgressCtrl wndProgress;
wndProgress.Create (WS_CHILD|WS_VISIBLE|WS_BORDER,
  CRect (x1, y1, x2, y2), this, IDC_PROGRESS);
```

头文件 Afxcmn.h 包含 CProgressCtrl 和其他公用控件类的声明。第二种方法是在对话框模板中添加一个 CONTROL 语句。建立对话框时,控件也随之建立。下面这个 CONTROL 语句在对话框中创建了一个进度控件:

```
CONTROL "", IDC_PROGRESS, PROGRESS_CLASS, WS_BORDER, 32, 32, 80, 16
```

用这种方法创建公用控件时,可以按照自己喜欢的方式或指定有字面意义的 WNDCLASS 名称,或指定一个别名。当用 Visual C++ 对话框编辑器给对话框添加公用控件时,系统会自动添加 CONTROL 语句。

大部分公用控件支持它们自己的窗口样式。这些样式可以和 WS_CHILD、WS_VISIBLE 和其他标准窗口样式混合起来一起使用。表 16-3 给出了一些"普通"公用控件样式,至少这些样式不专属于任一控件类型。作为 MFC 编程人员,很少直接操作这些样式,因为其中许多样式只用于工具栏和状态栏。如果用 CToolBar 和 CStatusBar 而不用较原始的 CToolBarCtrl 和 CStatusBarCtrl 类实现工具栏和状态栏,您就会得到较令人满意的 CCS 样式。表中这些样式并不是用于公用控件的所有样式。在介绍单个控件样式时,会特别介绍控件的专有样式。

表 16-3 公用控件样式

样式	说 明
CCS_TOP	将控件放在父窗口客户区的顶部,使控件宽度和父窗口宽度一致。工具栏以该样式作为默认样式
CCS_BOTTOM	将控件放在父窗口客户区的底部,使控件宽度和父窗口宽度一致。状态栏以该样式作为默认样式
CCS_LEFT *	将控件放在父窗口客户区的左端
CCS_RIGHT *	将控件放在父窗口客户区的右端
CCS_VERT *	竖直方向放置控件
CCS_NOMOVEX *	父窗口调整尺寸时,使控件只沿竖直方向调整大小并移动
CCS_NOMOVEY	父窗口调整尺寸时,使控件只沿水平方向调整大小并移动。标头控件以该样式作为默认样式
CCS_NORESIZE	禁止控件随父窗口的尺寸变化调整大小。如果指定了该样式,控件的宽和高就是控件矩形中指定的大小

续表

样式	说　明
CCS_NOPARENTALIGN	禁止控件呆在父窗口客户区的顶部或底部不动，该样式下的控件位置都是相对于父窗口客户区的左上角确定的。如果该样式和 CCS_TOP 或 CCS_BOTTOM 混合起来用，则控件就有了默认高度，并且宽度和位置不随父窗口的尺寸变化而改变
CCS_NODIVIDER	清除工具栏控件顶部的分隔栏
CCS_ADJUSTABLE	使工具栏控件中嵌入的自定义功能有效。双击该类工具栏，会显示一个 Customize Toolbar 对话框

* 需要 Internet Explorer 3.0 或更高版本

公用控件建立后，就可以使用相应控件类的成员函数操作它了。对于用对话框模板创建的控件，可以使用第 8 章介绍的任一种方法产生确定类型的引用，进而访问控件的函数和数据成员。例如：下列语句将 CProgressCtrl 的成员变量 m_wndProgress 和 ID 为 IDC_PROGRESS 的进度控件联系起来：

```
DDX_Control (pDX, IDC_PROGRESS, m_wndProgress);
```

对话框类的 DoDataExchange 函数中必须包含该语句。如果您喜欢，可以不必手工添加该语句，用 ClassWizard 就能做到。第 8 章介绍了如何使用 ClassWizard 将对话框类的成员变量和对话框中的控件联系起来。

在 SDK 风格的应用程序中使用公用控件时，必须在创建第一个控件之前调用 ::InitCommonControls 或它的新版本 ::InitCommonControlsEx，加载 Comctl32.dll 并注册控件的 WNDCLASS。在 MFC 应用程序中，MFC 替您调用这些函数。首先它调用 ::InitCommonControlsEx。如果系统没有安装 Internet Explorer 3.0 或更高版本（Internet Explorer 将 ::InitCommonControlsEx 添加到了 Win32 API）而导致调用失败，MFC 再调用 ::InitCommonControls。安装了 Windows 95 或更高版本，或 Windows NT 3.51 或更高版本的系统都支持该函数。

一旦创建了对话框，或调用了公用控件类的 Create 函数，MFC 都会调用 ::InitCommonControls(Ex)。如果因为某些原因您决定用 Windows API 创建公用控件或包含公用控件的对话框，或者决定用 CreateEx 而不是 Create 创建公用控件，您就要自己调用 ::InitCommonControls 或 ::InitCommonControlsEx。尽管您可以将调用推迟到控件或对话框建立之前，但调用最好在主窗口的 OnCreate 处理程序或 InitInstance 中进行。在应用程序生存期中多次调用 ::InitCommonControls(Ex) 不会产生负面影响。

16.1.2　处理通知：WM_NOTIFY 消息

典型控件利用 WM_COMMAND 消息把通知传送到父窗口。而和典型控件不同，大部分公用控件把通知封装在 WM_NOTIFY 消息中。WM_NOTIFY 消息的 wParam 保存发送消息控

件的了窗口 ID,而 lParam 保存指向 NMHDR 结构或 NMHDR 超集结构的指针。NMHDR 按如下方式定义:

```
typedef struct tagNMHDR {
    HWND hwndFrom;
    UINT idFrom;
    UINT code;
} NMHDR;
```

hwndFrom 保存控件的窗口句柄,idFrom 保存控件 ID(和 wParam 传送的值相同),而 code 指定通知代码。所有公用控件都传送表 16-4 中的通知。

表 16-4 公用控件传送的通知

通知	发送时间
NM_CLICK	鼠标左键单击控件
NM_DBLCLK	鼠标左键双击控件
NM_RCLICK	鼠标右键单击控件
NM_RDBLCLK	鼠标右键双击控件
NM_RETURN	控件具有输入焦点时按下 Enter 键
NM_KILLFOCUS	控件失去输入焦点
NM_SETFOCUS	控件获得输入焦点
NM_OUTOFMEMORY	由于内存不足,控件上某次操作失败

装有 Internet Explorer 3.0 或更高版本的系统可以支持更多种类的 NM 通知。例如:某些控件类型,包括一些初级公用控件,它们并不是绝对依赖 Internet Explorer 的支持,但是系统安装 Internet Explorer 后,这些控件的性能也增强了。这些控件发送 NM_CUSTOMDRAW 通知,使其所有者可以定义它们的外观。其他控件发送 NM_SETCURSOR 通知,使其所有者可以使用自定义光标。如果有的话,单个控件的文献会特别标注这些"特殊的"NM 通知。

大部分公用控件定义其他通知代码,表示控件特有的事件。例如:树形视图控件在展开子树时会通过发送 WM_NOTIFY 消息通知它的父窗口,其中 code 等价于 TVN_ITEMEXPANDED。lParam 指向一个 NM_TREEVIEW 结构,该结构包含以下数据成员:

```
typedef struct__NM_TREEVIEW {
  NMHDR    hdr;
  UINT     action;
  TV_ITEM  itemOld;
  TV_ITEM  itemNew;
  POINT    ptDrag;
} NM_TREEVIEW;
```

注意结构的第一个成员是一个 NMHDR 结构,使 NM_TREEVIEW 成为一个功能性的 NMHDR 超集。lParam 所指结构的类型取决于发送通知的控件的类型,有时甚至取决于通知代码。举例说明,树形视图控件发送 TVN_GETDISPINFO 通知时,lParam 指向一个TV_DISPINFO结

构，该结构的定义与 NM_TREEVIEW 大不相同：

```
typedef struct__TV_DISPINFO {
  NMHDR hdr;
  TV_ITEM item;
} TV_DISPINFO;
```

如何确定 lParam 中的指针类型呢？您先赋给它一个 NMHDR 指针，然后看看通知代码。这时，如果有必要，再更换一个更特殊的指针类型，如下所示：

```
NMHDR * pnmh = (NMHDR * ) lParam;
switch (pnmh -> code) {

case TVN_ITEMEXPANDED:
NM_TREEVIEW * pnmtv = (NM_TREEVIEW * ) pnmh;
// Process the notification.
break;

case TVN_GETDISPINFO:
NM_DISPINFO * pnmdi = (NM_DISPINFO * ) pnmh;
// Process the notification.
break;
}
```

如果处理这些通知的窗口包含两个或更多个树形视图控件，通过检查 NMHDR 结构的 hwndFrom 或 idFrom 字段，就可以识别发送通知的控件。

switch 语句和上面那个语句一样在 MFC 应用程序中都不是必要的，因为封装在 WM_NOTIFY 消息中的通知通过 ON_NOTIFY 和 ON_NOTIFY_RANGE 宏映射到类成员函数。此外，利用 ON_NOTIFY_REFLECT 可以将 WM_NOTIFY 通知反射回派生控件类。（MFC 还支持这些宏的扩展形式：ON_NOTIFY_EX、ON_NOTIFY_EX_RANGE 和 ON_NOTIFY_REFLECT_EX。）下列消息映射项将来自 ID 为 IDC_TREEVIEW 的树形视图控件的 TVN_ITEMEXPANDED 和 TVN_GETDISPINFO 通知映射为处理函数 OnItemExpanded 和 OnGetDispInfo：

```
ON_NOTIFY (TVN_ITEMEXPANDED, IDC_TREEVIEW, OnItemExpanded)
ON_NOTIFY (TVN_GETDISPINFO, IDC_TREEVIEW, OnGetDispInfo)
```

赋给 lParam 特殊指针类型是在通知处理程序中实现的：

```
void CMyWindow::OnItemExpanded (NMHDR * pnmh, LRESULT * pResult)
{
  NM_TREEVIEW * pnmtv = (NM_TREEVIEW * ) pnmh;
  // Process the notification.
}

void CMyWindow::OnGetDispInfo (NMHDR * pnmh, LRESULT * pResult)
```

```
{
  NM_DISPINFO * pnmdi = (NM_DISPINFO * ) pnmh;
  // Process the notification.
}
```

传递给 ON_NOTIFY 处理程序的参数 pnmh 和 WM_NOTIFY 消息中的 lParam 是一样的。参数 pResult 指向一个 32 位的 LRESULT 变量,该变量接受处理程序的返回值。许多通知的返回值没有任何意义,这种情况下处理程序忽略 pResult 没有任何影响。但是有时下面任务的执行和 * pResult 的值有直接关系。例如:您可以通过处理 TVN_ITEMEXPANDING 通知并把 * pResult 设置成非零值,来禁止展开树形视图控件的分支。另一方面,如果返回值是 0,则允许展开分支:

```
// In the message map
ON_NOTIFY (TVN_ITEMEXPANDING, IDC_TREEVIEW, OnItemExpanding)
  .
  .
  .
void OnItemExpanding (NMHDR * pnmh, LRESULT * pResult)
{
  NM_TREEVIEW * pnmtv = (NM_TREEVIEW * ) pnmh;
  if (...) {
    * pResult = TRUE; // Under certain conditions, prevent
    return;     // the expansion from taking place.
  }
  * pResult = 0;    // Allow the expansion to proceed.
}
```

TVN_ITEMEXPANDING 通知和 TVN_ITEMEXPANDED 通知不同,它在树形视图控件某项展开"之前"发送,而不是在那之后。对于标准控件类型,您可以忽略不感兴趣的通知,而只处理对应用程序有影响的通知。Windows 为没有处理过的通知提供适当的默认响应。

16.2 滑杆、微调按钮和工具提示控件

既然已经了解了公用控件的一般特性,下面我们就具体看看几种控件。我们先从滑杆、微调按钮和工具提示控件开始。这几个控件很容易编程实现,并且具有很强的通用性,可以用在各种应用程序中。在了解了这些控件和相应的 MFC 控件类之后,我们要编写一个示例程序。在该程序中要用到滑杆控件,用到工具提示控件提供与上下文有关的帮助,在程序对话框中还用到一对微调按钮。CToolTipCtrl 还不太成熟,我们不准备用它实现工具提示控件;这里,我们用 CToolTipCtrl 作为生成类的基类,并添加一对容易使用的成员函数。这对成员函数弥补了 MFC 实现工具提示控件中的严重缺陷。

16.2.1　滑杆控件

滑杆控件也称为“音轨栏控件”，它和收放机系统中的滑动式音量控制器相似。滑杆控件上有一个滑块，它的移动方式和滚动条滑块相似。滑杆控件创立后，您就可以设定一对最大值和最小值分别代表滑块的两个位置极限，以及滑块的初始位置。这时用户就可以通过用鼠标左键拖拉滑块，或单击滑块所在的通道调整滑块的位置。当滑杆拥有输入焦点时，移动滑块还可以用箭头键、Page Up 和 Page Down 键，以及 Home 和 End 键。只需调用一个函数就可以返回代表滑块位置的值。如果有必要，通过处理控件通知您可以响应滑块的位置变化。图 16-2 给出了一个简单的滑杆控件。刻度线表示滑块的允许位置。

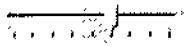

图 16-2　水平滑杆和表示滑块停止位的刻度线

表 16-5 给出滑杆特有的控件样式。滑杆控件可以沿水平或竖直方向放置。如果既没有指定 TBS_HORZ 也没有指定 TBS_VERT，则默认方向为水平方向。TBS_AUTOTICKS 样式用刻度线标识滑块停止位。如果滑杆的范围是 0 到 8，则 TBS_AUTOTICKS 设立 9 个刻度线——两个在滑杆两端，7 个在中间。TBS_NOTICKS 取消所有的刻度线，而 TBS_NOTHUMB 创建的滑杆没有滑块。如果您既没有指定 TBS_AUTOTICKS 也没有指定 TBS_NOTICKS，则创立的滑杆只在两端有刻度线，中间什么也没有。在默认方式下，刻度线画在水平滑杆的下方，竖直滑杆的右侧。通过指定 TBS_TOP 或 TBS_LEFT，可以把刻度线放在水平滑杆的上方或竖直滑杆的左侧，或者可以用 TBS_BOTH 创建滑杆，使滑杆上、下方或左、右侧都有刻度线。

表 16-5　滑杆控件样式

样式	说　明
TBS_HORZ	水平放置滑杆
TBS_VERT	竖直放置滑杆
TBS_LEFT	刻度线画在竖直滑杆的左侧
TBS_RIGHT	刻度线画在竖直滑杆的右侧
TBS_TOP	刻度线画在水平滑杆的上方
TBS_BOTTOM	刻度线画在水平滑杆的下方
TBS_BOTH	刻度线画在水平滑杆的上、下方或竖直滑杆的左、右侧
TBS_NOTICKS	清除滑杆的刻度线
TBS_AUTOTICKS	在滑杆范围内每个停止位处画一条刻度线
TBS_FIXEDLENGTH	允许给控件发送 TBM_SETTHUMBLENGTH 消息，改变滑块大小
TBS_NOTHUMB	清除滑杆的滑块
TBS_ENABLESELRANGE	加宽滑杆通道，显示选择范围
TBS_TOOLTIPS *	添加一个动态工具提示控件，该控件随滑块移动并显示滑块位置。通过 CSliderCtrl::SetToolTips 把默认工具提示控件换成自己的控件

* 需要 Internet Explorer 3.0 或更高版本

MFC 用 CSliderCtrl 类代表滑杆。滑杆的范围和滑块位置用 CSliderCtrl::SetRange 和 CSliderCtrl::SetPos 设定。相关的函数 CSliderCtrl::GetRange 和 CSliderCtrl::GetPos 提取范围和位置信息。如果 m_wndSlider 是一个 CSliderCtrl 类,语句

```
m_wndSlider.SetRange (0, 8);
m_wndSlider.SetPos (2);
```

将滑杆范围设置成 0 到 8,滑块位置设置成 2。

样式为 TBS_AUTOTICKS 的滑杆控件在滑块每个位置增量上画一个刻度线。可以用 CSliderCtrl::SetTicFreq 调整刻度线间的距离。下列语句使滑杆控件每隔一个滑块停止位画一条刻度线:

```
m_wndSlider.SetTicFreq (2);
```

如果要使滑杆获得不等间距的刻度线,则删除样式 TBS_AUTOTICKS,并用 CSliderCtrl::SetTic 把刻度线放在要求的位置上。除了 0、8 位置上的刻度线,语句

```
m_wndSlider.SetRange (0, 8);
m_wndSlider.SetTic (2);
m_wndSlider.SetTic (3);
m_wndSlider.SetTic (6);
m_wndSlider.SetPos (2);
```

将刻度线添加在 2、3 和 6 位置。

TBS_ENABLESELRANGE 样式创建了一个通道较宽的滑杆,适合显示选择范围。选择范围由 CSliderCtrl::SetSelection 设定,并用颜色为系统颜色(COLOR_HIGHLIGHT)的棒形图表示。语句

```
m_wndSlider.SetRange (0, 8);
m_wndSlider.SetSelection (3, 7);
```

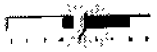

图 16-3 具有选择范围的滑杆

将范围设成 0 到 8,选择范围设成 3 到 7。生成的滑杆请见图 16-3。设置选择范围并不影响滑块的行程,滑块仍旧可以放在滑杆范围内的任意位置上。如果需要把滑块行程限制在选择范围内,或允许用户改变选择范围,则必须使用自定义滑杆控件 UI。最实用的 UI 自定义方法是:从CSliderCtrl 派生出一个类,并添加消息处理程序,改变控件对按下 Home、End、Page Up、Page Down 和箭头键以及单击鼠标左键的响应方式。如果要对选中消息执行默认处理程序,则只需把消息传递给基类。

移动滑块时,和滚动条一样,滑杆向它的父窗口发送 WM_HSCROLL 或 WM_VSCROLL 消息。滑杆控件的 OnHScroll 或 OnVScroll 处理程序接收三个参数:一个通知代码、一个整型数(指定滑块最新位置)和一个 CScrollBar 指针(该指针可以被强制转换为 CSliderCtrl 指针)。表 16-6 给出了 9 个可能会遇到的通知代码和引发它们的动作。传递给 OnHScroll 或

OnVScroll的滑块位置只有在通知代码为 TB_THUMBPOSITION 或 TB_THUMBTRACK 的时候有效。如果要响应其他类型的通知，则用 CSliderCtrl::GetPos 提取滑块位置信息。

表 16-6　滑杆通知

通知	发送时间
TB_TOP	滑杆具有输入焦点时按下 Home 键
TB_BOTTOM	滑杆具有输入焦点时按下 End 键
TB_LINEDOWN	滑杆具有输入焦点时按下向下或向右箭头键
TB_LINEUP	滑杆具有输入焦点时按下向上或向左箭头键
TB_PAGEDOWN	滑杆具有输入焦点时按下 Page Down 键，或在水平滑杆中单击滑块右侧的通道，或在竖直滑杆中单击滑块下方的通道
TB_PAGEUP	滑杆具有输入焦点时按下 Page Up 键，或在水平滑杆中单击滑块左侧的通道，或在竖直滑杆中单击滑块上方的通道
TB_THUMBTRACK	用鼠标把滑块拖放到新位置
TB_THUMBPOSITION	滑块拖放后释放鼠标左键
TB_ENDTRACK	释放移动滑块的键或鼠标键

滑杆通知的一个用途是：响应位置变化，动态更新屏幕上的图像。通过右击桌面并在上下文菜单中选择 Properties，可以打开系统的 Display Properties 属性表，其中 Settings 页会处理 Screen Area 框中滑杆发送的 TB_THUMBTRACK 通知，并随滑块的每次移动重画计算机图像，使您可以预览新设置对桌面产生的影响。

CSliderCtrl 提供了二十余种操作滑杆控件的函数。其他有用的成员函数包括：SetPageSize，它确定单击杆体或按下 Page Up 或 Page Down 键时滑块移动的单位数；GetTic、GetTicPos、GetTicArray 和 GetNumTicks，它们返回刻度线的信息；以及 ClearSel，它清除选中范围。欲知 CSliderCtrl 函数成员的详细信息，请见 MFC 文档说明。

16.2.2　微调按钮控件

微调按钮控件，也称为“上下控件”，是包含向上和向下或向左和向右箭头的小窗口。和滚动条与滑杆相似，微调按钮有它们自己的范围和位置。单击向上或向右箭头，当前位置会向前走；单击向下或向左箭头，当前位置会向后退。微调按钮控件在位置每次变化之前和之后都给它们的父窗口发送消息，但是因为微调按钮可以独立完成一些非常有价值的工作，所以这些通知经常被忽略。

在创建微调按钮控件时，您可以在表 16-7 中选择样式。UDS_SETBUDDYINT 创建的微调按钮控件可以自动更新显示在“伙伴”控件中的整型值。“伙伴”控件通常是编辑控件或静态文本控件。UDS_SETBUDDYINT 样式的微调按钮控件发生位置变化时，它将描述新位置的整型值转换为文本字符串（如通过_itoa），并使用 ::SetWindowText 将字符串显示在“伙伴”控件中。UDS_SETBUDDYINT 使在编辑控件上添加箭头变得简单。因此，用户既可以用

键盘敲入数字,也可以用鼠标拨号。

表 16-7 微调按钮控件样式

样式	说 明
UDS_HORZ	使箭头呈水平方向,而不是竖直方向
UDS_WRAP	如果前进或后退超出了最大值或最小值,则位置按环式结构确定
UDS_ARROWKEYS	添加键盘接口。如果这种样式的微调按钮控件获得输入焦点,则向上和向下箭头可以向前和向后挪移焦点
UDS_NOTHOUSANDS	清除千位分隔符,即:使 1 234 567 显示为 1234567
UDS_SETBUDDYINT	创建微调按钮控件,使它在位置变化时更新伙伴控件中的文本
UDS_AUTOBUDDY	按照 Z 向顺序选中以前的控件作微调按钮的伙伴
UDS_ALIGNRIGHT	将微调按钮控件填在伙伴控件内的右边缘
UDS_ALIGNLEFT	将微调按钮控件填在伙伴控件内的左边缘

可以用两种方法将微调按钮控件和它的伙伴控件联系起来。通过指向伙伴控件的 CWnd 指针调用 CSpinButtonCtrl::SetBuddy,您可以显式连接两者;或者,在创建微调按钮控件时指定 UDS_AUTOBUDDY,则该微调按钮控件自动按照 Z 向顺序选中以前的控件作它的伙伴。在对话框模板中,语句

```
EDITTEXT    IDC_EDIT, 60, 80, 40, 14, ES_AUTOHSCROLL
CONTROL     "", IDC_SPIN, "msctls_updown32", UDS_SETBUDDYINT |
    UDS_AUTOBUDDY|UDS_ALIGNRIGHT, 0, 0, 0, 0
```

创建了一个单行编辑控件,并将微调按钮控件填在它的右边缘的内侧,如图 16-4 所示。编辑控件根据微调按钮控件的宽度收缩,而微调按钮的高度调整到伙伴控件的高度。因此,编辑控件和微调按钮控件占据的是编辑控件原来占据的空间。如果指定了 UDS_ALIGNLEFT 或 UDS_ALIGNRIGHT,微调按钮控件的尺寸和位置信息则被忽略。

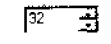

图 16-4 附在编辑控件上的微调按钮控件

在默认方式下,UDS_SETBUDDYINT 微调按钮控件以十进制显示数字,并每隔三位插入一个千位分隔符。可以设置控件,而不必使用 CSpinButtonCtrl::SetBase,使控件显示十六进制数字:

```
m_wndSpinButton.SetBase (16);
```

十六进制数字的前面都加上 0x。这样,一看就知道是十六进制的。通过 10 调用 SetBase 可以把输出切换为十进制格式。通过在创建控件时指定 UDS_NOTHOUSANDS,可以清除十进制数字中的分隔符;默认方式下省略十六进制数字中的千位分隔符。

运用 CSpinButtonCtrl::SetRange 和 CSpinButtonCtrl::SetPos 设置微调按钮控件的范围和位

置。最大和最小值的有效范围是从 - 32 767 到 32 767,但是最小和最大值间的差别不能超过 32 767。指定最大值小于最小值是合法的。如果这样设定了,则各箭头的动作就反过来了。如果系统安装了 Internet Explorer 4.0 或更高版本,则微调按钮控件支持 32 位范围,其最大和最小值可以用 CSliderCtrl 函数 SetRange32 和 GetRange32 设定和提取。

在微调按钮控件中每单击一次箭头(在控件样式包括 UDS_ARROWKEYS 时则可以按箭头键),位置都会前进或后退一个。如果按住按钮,则 2 秒后增量变成 ±5,5 秒后增量变成 ±20。可以改变增量变化前需要的秒数,还可以用 CSpinButtonCtrl::SetAccel 控制变化的大小。SetAccel 接受两个参数:指向 UDACCEL 结构数组的指针和数组中的结构数。下列语句将微调按钮控件设置为:按钮按下时,前两秒位置增量为 1,下一个 2 秒为 2,再下一个 2 秒为 10,时间再长,增量就变为 100 :

```
UDACCEL uda[4];
uda[0].nSec = 0;
uda[0].nInc = 1;
uda[1].nSec = 2;
uda[1].nInc = 2;
uda[2].nSec = 4;
uda[2].nInc = 10;
uda[3].nSec = 8;
uda[3].nInc = 100;
pSpinButton -> SetAccel (4, uda);
```

SetAccel 的另一个用处是指定增量为 1 之外的其他值。如果您希望每单击一次按钮,位置就前进或后退 5,则要这样调用 SetAccel:

```
UDACCEL uda;
uda.nSec = 0;
uda.nInc = 5;
pSpinButton -> SetAccel (1, &uda);
```

通过将 UDACCEL 结构数组的地址传递给 CSpinButton::GetAccel,可以获取加速值。但这里有一个技巧:怎么知道要给多少个结构分配空间呢?在 Visual C++ 6 之前这个问题没有得到解决,但按照以下方式调用 GetAccel,可以返回加速器数组中 UDACCEL 结构的数目:

```
UINT nCount = pSpinButton -> GetAccel (0, NULL);
```

一旦知道了数目,您就可以为数组分配缓冲区并提取数组,如下所示:

```
UDACCEL * puda = new UDACCEL[nCount];
pSpinButton -> GetAccel (nCount, puda);
// Do something with the array.
delete[] puda;
```

明白了吗？当您知道这个技巧后，问题就很简单了。

在位置变化之前，微调按钮控件向它的父窗口发送 WM_NOTIFY 消息，同时还带有等于 UDN_DELTAPOS 的通知代码和指向 NM_UPDOWN 结构的 lParam 指针。该结构中有些整型数，如确定当前位置（iPos）和位置增量的整型值（iDelta）。UDN_DELTAPOS 处理程序必须把 * pResult 设成 FALSE，允许位置变化发生。如果有意要阻止位置变化的发生，则使处理程序将 * pResult 设成 TRUE，而后由 OnNotify 返回 TRUE。跟在 UDN_DELTAPOS 通知后面的是 WM_HSCROLL 或 WM_VSCROLL 消息（取决于微调按钮是水平方向的还是竖直方向的），它们报告新位置。控件当前位置为 8 时，单击向下箭头，则产生表 16-8 中的消息。

表 16-8 产生的消息

消息	通知代码	参数
WM_NOTIFY	UDN_DELTAPOS	iPos = 8, iDelta = -1
WM_VSCROLL	SB_THUMBPOSITION	nPos = 7
WM_VSCROLL	SB_ENDSCROLL	nPos = 7

如果按钮按下达半秒多时，则有 UDN_DELTAPOS 和 SB_THUMBPOSITION 通知先后被发送。

16.2.3 工具提示控件

工具提示是一个小型帮助文本窗口。当光标停在工具栏按钮或对话框控件等“工具”上时，这个小型窗口就会出现在屏幕上。工具提示控件可以监视鼠标移动，并当光标在某工具上停留预定时间后自动显示工具提示。MFC 通过 CToolTipCtrl 类给工具提示控件提供了一个方便的 C++ 接口。有了 CToolTipCtrl，给对话框中控件添加工具提示，以及实现其他形式的交互帮助就变得相对容易了。您可以轻松地创建工具提示控件，注册服务对象和工具提示文本。其他的工作由控件来完成。

CToolTipCtrl::Create 创建了一个工具提示控件。（可以由对话框模板创建工具提示控件，但是更常见的方法是在对话框类中添加 CToolTipCtrl 数据成员，并在 OnInitDialog 中调用 Create。）如果 m_ctlTT 是窗口类的 CToolTipCtrl 数据成员，则语句

```
m_ctlTT.Create(this);
```

创建了一个工具提示控件。CToolTipCtrl::Create 还接受第二个可选参数，该参数指定控件的样式。系统只支持两种样式 TTS_ALWAYSTIP 和 TTS_NOPREFIX。默认方式下，工具提示只出现在有效窗口中。TTS_ALWAYSTIP 样式的工具提示控件则可以在有效和无效窗口中同时显示工具提示。TTS_NOPREFIX 通知控件不能去除工具提示文本中的 & 符号。默认方式是忽略 & 符号，则同一个文本字符串既可以用于菜单，又可以用于工具提示。

工具提示控件创立后，下一步就是在其中添加工具。工具既可以是另外一个窗口（通常

是个子窗口，属于工具提示控件的父窗口），也可以是窗口的一个矩形域。CToolTipCtrl::AddTool注册工具以及相关的工具提示文本。一个工具提示控件可以连接任意个工具。语句

```
m_ctlTT.AddTool (pWnd, __T ("This is a window"), NULL, 0);
```

将工具提示文本"This is a window"赋给 pWnd 指定的窗口。传递给 AddTool 的第二个参数可以是指向文本字符串的指针，也可以是字符串资源的 ID。随便用哪个都行。语句

```
m_ctlTT.AddTool (pWnd, __T ("This is a rectangle"),
CRect (32, 32, 64, 64), IDT_RECTANGLE);
```

在 pWnd 用户区的指定矩形中创建了一个工具。IDT_RECTANGLE 是一个非零的整型值，指定该矩形。这和子窗口 ID 指定控件类似。

到目前为止，一切都很顺利。只有一个问题。工具提示控件必须得到工具接收到的鼠标消息，这样，它才能监视鼠标事件，知道什么时候该显示工具提示。但事实是：Windows 把鼠标消息发送给了光标下的窗口。在上面的示例中，您必须把送往 pWnd 的消息转向工具提示控件。如果 pWnd 是一个顶层窗口或对话框，转送鼠标消息并不困难。因为可以把相关鼠标消息映射为窗口类或对话框类的处理程序，并通过 CToolTipCtrl::RelayEvent 把消息转送到工具提示控件。但是如果 pWnd 指向一个子窗口控件或任何其他窗口，为了能得到送往窗口的鼠标消息并将它们转送到工具提示控件，就不得不求助于窗口子类划分或其他手段。在后来的 Windows 95 β 测试版中，操作系统设计者认识到这个问题，并给予工具提示控件实现自己的子类划分的能力。不幸的是，不得不把该功能限制在 CToolTipCtrl 中。因此，为了使工具提示确实好用，必须添加一些自己的特色，自定义 CToolTipCtrl 类。

只要我在 MFC 应用程序中使用工具提示控件，我都要先从 CToolTipCtrl 派生出一个类 CMyToolTipCtrl 并添加一对成员函数。其中成员函数正是利用了工具提示控件可以实现自己的子类划分这一点。派生类如下所示：

```
class CMyToolTipCtrl : public CToolTipCtrl
{
public:
  BOOL AddWindowTool (CWnd * pWnd, LPCTSTR pszText);
  BOOL AddRectTool (CWnd * pWnd, LPCTSTR pszText,
    LPCRECT pRect, UINT nIDTool);
};

BOOL CMyToolTipCtrl::AddWindowTool (CWnd * pWnd, LPCTSTR pszText)
{
  TOOLINFO ti;
  ti.cbSize = sizeof (TOOLINFO);
  ti.uFlags = TTF_IDISHWND TTF_SUBCLASS;
  ti.hwnd = pWnd -> GetParent () -> GetSafeHwnd ();
```

```
    ti.uId = (UINT) pWnd->GetSafeHwnd();
    ti.hinst = AfxGetInstanceHandle();
    ti.lpszText = (LPTSTR) pszText;

    return (BOOL) SendMessage(TTM_ADDTOOL, 0, (LPARAM) &ti);
}

BOOL CMyToolTipCtrl::AddRectTool(CWnd* pWnd, LPCTSTR pszText,
    LPCRECT lpRect, UINT nIDTool)
{
    TOOLINFO ti;
    ti.cbSize = sizeof(TOOLINFO);
    ti.uFlags = TTF_SUBCLASS;
    ti.hwnd = pWnd->GetSafeHwnd();
    ti.uId = nIDTool;
    ti.hinst = AfxGetInstanceHandle();
    ti.lpszText = (LPTSTR) pszText;
    ::CopyRect(&ti.rect, lpRect);

    return (BOOL) SendMessage(TTM_ADDTOOL, 0, (LPARAM) &ti);
}
```

有了这个基础,在子窗口控件中创建工具(子类划分等等)只需要一个简单语句:

```
m_ctlTT.AddWindowTool(pWnd,_T("This is a window"));
```

在窗口矩形中创建工具也同样简单:

```
m_ctlTT.AddRectTool(pWnd,_T("This is a rectangle"),
CRect(32, 32, 64, 64), IDT_RECTANGLE);
```

传递给 AddWindowTool 的 pWnd 参数确定了工具提示所在的窗口。传递给 AddRectTool 的 pWnd 参数引用客户区包含矩形的窗口。该矩形则在第三个参数中被引用。因为有了在 TOOLINFO 结构的 uFlags 域中传送的 TTF_SUBCLASS 标志,工具提示控件就会实现自己窗口的子类划分,鼠标消息就不必通过手工编程转送了。

动态工具提示

如果在调用 AddTool、AddWindowTool 或 AddRectTool 时,给工具提示文本指定了 LPSTR_TEXTCALLBACK,工具提示控件就会在显示工具提示之前向它的父窗口发送一个通知,请求文本字符串。可以用 LPSTR_TEXTCALLBACK 创建动态工具提示,使这次请求和下次请求得到的文本都不相同。得到的文本以 WM_NOTIFY 消息的形式出现。其中它的通知代码为 TTN_NEEDTEXT, lParam 指向类型为 TOOLTIPTEXT 的结构。TOOLTIPTEXT 以如下形式定义:

```
typedef struct {
  NMHDR   hdr;
  LPTSTR  lpszText;
  char    szText[80];
  HINSTANCE hinst;
  UINT    uFlags;
} TOOLTIPTEXT;
```

ToolTip 控件的父窗口可以用三种方式响应 TTN_NEEDTEXT 通知，或者把文本字符串的地址复制到 TOOLTIPTEXT 结构的 lpszText 域中；或者把文本（可以有 80 个字符，包括零停止位）直接复制到结构的 szText 域；或者把字符串资源 ID 复制到 lpszText 并把应用程序的实例句柄（MFC 应用程序可以通过 AfxGetInstanceHandle 获得）复制到 hinst。NMHDR 结构（嵌套在 TOOLTIPTEXT 结构中）的 idFrom 域可以包含窗口句柄，也可以包含应用程序定义的工具 ID（指定需要帮助文本的工具）。

下面的示例说明如何在对话框矩形区中创建动态工具提示，该矩形的工具 ID 为 IDT_RECTANGLE，工具提示窗口中显示的文本是当前时间：

```
// In the message map
ON_NOTIFY (TTN_NEEDTEXT, NULL, OnNeedText)
  .
  .
  .
BOOL CMyDialog::OnInitDialog ()
{
  m_ctlTT.Create (this);
  m_ctlTT.AddRectTool (this, LPSTR_TEXTCALLBACK,
    CRect (0, 0, 32, 32), IDT_RECTANGLE);
  return TRUE;
}

void CMyDialog::OnNeedText (NMHDR * pnmh, LRESULT * pResult)
{
  TOOLTIPTEXT * pttt = (TOOLTIPTEXT *) pnmh;
  if (pttt->hdr.idFrom == IDT_RECTANGLE) {
    CString string;
    CTime time = CTime::GetCurrentTime ();
    string.Format (_T ("%0.2d:%0.2d:%0.2d"), time.GetHour () % 12,
      time.GetMinute (), time.GetSecond ());
    ::lstrcpy (pttt->szText, (LPCTSTR) string);
  }
}
```

注意，在 ON_NOTIFY 宏（在 CmyDialog 的消息映射表中）的第二个参数中指定的子窗口

ID 为 NULL。这个参数必须是 NULL,因为 CToolTipCtrl::Create 为工具提示控件注册的是 NULL 子窗口 ID。

MFC 的 CToolTipCtrl 类包含可以用来操作工具提示控件的一系列成员函数。例如:可以用 GetText 获取分配给工具的文本;可以用 UpdateTipText 改变工具提示文本;可以用 Activate激活或使工具提示控件失效;以及可以用 SetDelayTime 改变停滞时间(在工具提示显示之前,光标必须保持静止的时间。)。默认停滞时间为 500 毫秒。

16.2.4 GridDemo 应用程序

GridDemo 应用程序(源代码见图 16-6)使用了滑杆控件、微调按钮控件和工具提示控件。GridDemo 通过画彼此相交的水平线和垂直线把框架窗口的客户区划分成网格。在默认方式下,网格包含 8 行、8 列,并且网格线用中等程度的灰色画出。通过在 Options 菜单中选择 Grid Settings,并在图16-5所示的对话框中输入新设置,您可以改变行列数以及网格线的深浅。滑杆控件选择线的磅值,输入编辑控件的值确定行数和列数。有效值的范围是 2 到 64;可以敲入数字或使用箭头按钮。当光标停在滑杆或某个编辑控件上时,带有简短工具描述的工具提示窗口出现在该工具下面。

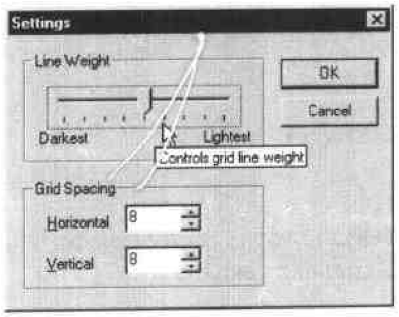

图 16-5 显示有工具提示的 GridDemo 的 Settings 对话框

ChildView.h

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

#if !defined(
  AFX_CHILDVIEW_H__A4559BAA_ABE5_11D2_8E53_006008A82731__INCLUDED_)
```

```
#define AFX_CHILDVIEW_H__A4559BAA_ABE5_11D2_8E53_006008A82731__INCLUDED_

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

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

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

// Attributes
public:

// Operations
public:

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

// Implementation
public:
  virtual CChildView();

  // Generated message map functions
protected:
  int m_nWeight;
  int m_cy;
  int m_cx;
  //{{AFX_MSG(CChildView)
  afx_msg void OnPaint();
  afx_msg void OnOptionsGridSettings();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif
// !defined(
//   AFX_CHILDVIEW_H__A4559BAA_A3E5_11D3_8E53_006008A82731__INCLUDED_)
```

ChildView.cpp

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

#include "stdafx.h"
#include "GridDemo.h"
#include "ChildView.h"
#include "SettingsDialog.h"

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

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

CChildView::CChildView()
{
  m_cx = 8;
  m_cy = 8;
  m_nWeight = 4;
}

CChildView::~CChildView()
{
}

BEGIN_MESSAGE_MAP(CChildView,CWnd)
  //{{AFX_MSG_MAP(CChildView)
  ON_WM_PAINT()
  ON_COMMAND(ID_OPTIONS_GRID_SETTINGS, OnOptionsGridSettings)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

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

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

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

  return TRUE;
}

void CChildView::OnPaint()
{
  CRect rect;
  GetClientRect(&rect);

  int nShade = m_nWeight * 32;
  if (nShade != 0)
    nShade--;

  CPaintDC dc(this);
  CPen pen(PS_SOLID, 1, RGB(nShade, nShade, nShade));
  CPen* pOldPen = dc.SelectObject(&pen);

  int x;
  for (int i=1; i<m_cx; i++) {
    x = (rect.Width() * i) / m_cx;
    dc.MoveTo(x, 0);
    dc.LineTo(x, rect.Height());
  }

  int y;
  for (i=1; i<m_cy; i++) {
    y = (rect.Height() * i) / m_cy;
    dc.MoveTo(0, y);
    dc.LineTo(rect.Width(), y);
  }

  dc.SelectObject(pOldPen);


void CChildView::OnOptionsGridSettings()
```

```
}
  CSettingsDialog dlg;

  dlg.m_cx = m_cx;
  dlg.m_cy = m_cy;
  dlg.m_nWeight = m_nWeight;

  if (dlg.DoModal () == IDOK) {
    m_cx = dlg.m_cx;
    m_cy = dlg.m_cy;
    m_nWeight = dlg.m_nWeight;
    Invalidate ();
  }
}
```

SettingsDialog.h

```
# if !defined(
  AFX_SETTINGSDIALOG_H__A4559BB0_ABE5_11D2_8E53_006008A82731__INCLUDED_)
# define
  AFX_SETTINGSDIALOG_H__A4559BB0_ABE5_11D2_8E53_006008A82731__INCLUDED_

# include "MyToolTipCtrl.h"  // Added by ClassView
# if_MSC_VER > 1000
# pragma once
# endif //_MSC_VER > 1000
// SettingsDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSettingsDialog dialog

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

// Dialog Data
  //{{AFX_DATA(CSettingsDialog)
  enum { IDD = IDD_SETTINGDLG };

  CSpinButtonCtrl  m_wndSpinVert;
  CSpinButtonCtrl  m_wndSpinHorz;
```

```
    CSliderCtrl  m_wndSlider;
    int    m_cx;
    int    m_cy;
    //}}AFX_DATA

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

// Implementation
protected:
    CMyToolTipCtrl m_ctlTT;
    // Generated message map functions
    //{{AFX_MSG(CSettingsDialog)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

#endif
// !defined(
// AFX_SETTINGSDIALOG_H__A4559BB0_ABE5_11D2_8E53_006008A82731__INCLUDED_)
```

SettingsDialog.cpp

```
// SettingsDialog.cpp : implementation file
//

#include "stdafx.h"
#include "GridDemo.h"
#include "MyToolTipCtrl.h"
#include "SettingsDialog.h"

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

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

/////////////////////////////////////////////////////////////////////////////
// CSettingsDialog dialog

CSettingsDialog::CSettingsDialog(CWnd* pParent /* = NULL */)
  : CDialog(CSettingsDialog::IDD, pParent)
{
  //{{AFX_DATA_INIT(CSettingsDialog)
  m_cx = 0;
  m_cy = 0;
  //}}AFX_DATA_INIT
}

void CSettingsDialog::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CSettingsDialog)
  DDX_Control(pDX, IDC_SPINVERT, m_wndSpinVert);
  DDX_Control(pDX, IDC_SPINHORZ, m_wndSpinHorz);
  DDX_Control(pDX, IDC_SLIDER, m_wndSlider);
  DDX_Text(pDX, IDC_EDITHORZ, m_cx);
  DDX_Text(pDX, IDC_EDITVERT, m_cy);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSettingsDialog, CDialog)
  //{{AFX_MSG_MAP(CSettingsDialog)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSettingsDialog message handlers

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

  //
  // Initialize the slider control.
  //

  m_wndSlider.SetRange (0, 8);
  m_wndSlider.SetPos (m_nWeight);
```

```
    //
    // Initialize the spin button controls.
    //
    m_wndSpinHorz.SetRange (2, 64);
    m_wndSpinVert.SetRange (2, 64);

    //
    // Create and initialize a tooltip control.
    //
    m_ctlTT.Create (this);
    m_ctlTT.AddWindowTool (GetDlgItem (IDC_SLIDER),
      MAKEINTRESOURCE (IDS_SLIDER));
    m_ctlTT.AddWindowTool (GetDlgItem (IDC_EDITHORZ),
      MAKEINTRESOURCE (IDS_EDITHORZ));
    m_ctlTT.AddWindowTool (GetDlgItem (IDC_EDITVERT),
      MAKEINTRESOURCE (IDS_EDITVERT));
    return TRUE;
}

void CSettingsDialog::OnOK()
{
    //
    // Read the slider control's thumb position
    // before dismissing the dialog.
    //
    m_nWeight = m_wndSlider.GetPos ();
    CDialog::OnOK();
}
```

MyToolTipCtrl.h

```
#if !defined(
  AFX_MYTOOLTIPCTRL_H__A4559BB1_A5E5_11D2_8E53_006008A82731__INCLUDED_ )
#define
  AFX_MYTOOLTIPCTRL_H__A4559BB1_A5E5_11D2_8E53_006008A82731__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MyToolTipCtrl.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CMyToolTipCtrl window

class CMyToolTipCtrl : public CToolTipCtrl
```

```
|
// Construction
public:
  CMyToolTipCtrl();

// Attributes
public:

// Operations
public:

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

// Implementation
public:
  BOOL AddRectTool (CWnd * pWnd, LPCTSTR pszText, LPCRECT pRect,
    UINT nIDTool);
  BOOL AddWindowTool (CWnd * pWnd, LPCTSTR pszText);
  virtual ~CMyToolTipCtrl();

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

  DECLARE_MESSAGE_MAP()
};

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

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

#endif
// !defined(
//  AFX_MYTOOLTIPCTRL_H__A4559BB1_ABE5_11D2_8E53_006008A82731__INCLUDED )
```

MyToolTipCtrl.cpp

```
// MyToolTipCtrl.cpp : implementation file
//

#include "stdafx.h"
```

```
#include "GridDemo.h"
#include "MyToolTipCtrl.h"

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

/////////////////////////////////////////////////////////////////////////////
// CMyToolTipCtrl

CMyToolTipCtrl::CMyToolTipCtrl()
{
}

CMyToolTipCtrl::~CMyToolTipCtrl()
{
}

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

/////////////////////////////////////////////////////////////////////////////
// CMyToolTipCtrl message handlers

BOOL CMyToolTipCtrl::AddWindowTool(CWnd * pWnd, LPCTSTR pszText)
{


  TOOLINFO ti;
  ti.cbSize = sizeof (TOOLINFO);
  ti.uFlags = TTF_IDISHWND | TTF_SUBCLASS;
  ti.hwnd = pWnd->GetParent ()->GetSafeHwnd ();
  ti.uId = (UINT) pWnd->GetSafeHwnd ();
  ti.hinst = AfxGetInstanceHandle ();
  ti.lpszText = (LPTSTR) pszText;

  return (BOOL) SendMessage (TTM_ADDTOOL, 0, (LPARAM) &ti);
}

BOOL CMyToolTipCtrl::AddRectTool(CWnd * pWnd, LPCTSTR pszText,
  LPCRECT pRect, UINT nIDTool)
{
  TOOLINFO ti;
```

```
    ti.cbSize = sizeof (TOOLINFO);
    ti.uFlags = TTF_SUBCLASS;
    ti.hwnd = pWnd->GetSafeHwnd ();
    ti.uId = nIDTool;
    ti.hinst = AfxGetInstanceHandle ();
    ti.lpszText = (LPTSTR) pszText;
    ::CopyRect (&ti.rect, pRect);

    return (BOOL) SendMessage (TTM_ADDTOOL, 0, (LPARAM) &ti);
}
```

图 16-6 GridDemo 应用程序

工具提示控件是 CMyToolTipCtrl 的一个实例。这里我没有费力地把工具提示文本编写到 AddWindowTool 的调用中,我选择把文本放在应用程序的字符串表中。字符串资源由它们的资源 ID 确定。在 AddWindowTool 的调用中,IDS_SLIDER、IDS_EDITHORZ 和 IDS_EDITVERT都是资源 ID:

```
m_ctlTT.AddWindowTool (GetDlgItem (IDC_SLIDER),
  MAKEINTRESOURCE (IDS_SLIDER));
m_ctlTT.AddWindowTool (GetDlgItem (IDC_EDITHORZ),
  MAKEINTRESOURCE (IDS_EDITHORZ));
m_ctlTT.AddWindowTool (GetDlgItem (IDC_EDITVERT),
  MAKEINTRESOURCE (IDS_EDITVERT));
```

打开项目并切换到 ResourceView,在字符串表中您可以看到和这些资源 ID 关联的文本。

滑杆和微调按钮控件是对话框模板的一部分,并使用 CSliderCtrl 和 CSpinButtonCtrl 成员函数编程得到。滑杆的范围和初始位置在 OnInitDialog 中设定,最终的滑块位置在 OnOK 中获取。微调按钮的范围也在 OnInitDialog 中初始化,但是因为编辑控件(微调按钮的伙伴控件),使用的是 Dialog Data Exchange (DDX) 和 Dialog Data Validation (DDV) 例程,所以微调按钮的位置不必显式设定或提取。

谈到 DDX 和 DDV:对于某些特殊情况,MFC 不提供 DDX 例程在公用控件和对话框数据成员间传送数据,或者不用 DDV 例程查验在公用控件中的输入。如果在对话框中只使用典型控件,则不必总要覆盖 OnInitDialog 和 OnOK。因为您(或 ClassWizard)可以用在对话框成员变量和它的控件间传送数据的语句填充 DoDataExchange。然而,如果使用公用控件,则您必须自己初始化控件并执行数据交换。这就是 CSettingsDialog::OnInitDialog 包含下面这些语句的原因:

```
m_wndSlider.SetRange (0, 8);
m_wndSlider.SetPos (m_nWeight);
```

```
    .
    .
    .
m_wndSpinHorz.SetRange (2, 64);
m_wndSpinVert.SetRange (2, 64);
```

并且 CSettingsDialog::OnOK 还包含语句：

```
m_nWeight = m_wndSlider.GetPos ();
```

这些语句完成了应该由 DDX 做(如果它得到支持的话)的工作。(非常有趣，MFC 6 包含了 DDX_Slider 函数，该函数可以在滑杆控件上执行 DDX；但是因为滑杆初始化时它只设定位置，而不设定范围，所以这个函数是残缺不可用的。试试看，您就会明白我的意思。) m_wndSlider 是 CSliderCtrl 成员变量，我通过 ClassWizard 把它添加到对话框类中。m_wndSpinHorz 和 m_wndSpinVert 是 CSpinButtonCtrl 成员变量，它们也是用 ClassWizard 添加到对话框类中的。三个成员变量都是通过 DoDataExchange 中的 DDX_Control 语句与控件相关联的。

因为 GridDemo 没有创建逻辑调色板(其中各种深浅的灰色代表不同的磅值设置)，所以在 16 色和 256 色图像适配器中看不到整个范围的磅值。作为一次练习，可以试着在框架窗口中添加 CPalette 数据成员，并使用 PALETTERGB 或 PALETTEINDEX 色绘制网格线，从而最终添加上调色板支持。有关 GDI 调色板和 MFC CPalette 类的详细信息，请参看第 15 章。

16.3　图像列表和 ComboBoxEx 控件

第 10 章的 DriveTree 和 WinDir 程序使用了图像列表给树形视图和列表视图提供图标图像。当时，除了谈到图像列表包含位图图像集合，以及 MFC 把它们封装在类 CImageList 中外，并没有涉及其他。实际上，图像列表特别有用，它不仅能为其他控件提供图像，而且还能在位图上造出特殊效果，如透明和混合效果。下面我们就详细介绍图像列表的功能。

如果安装了 Internet Explorer 3.0 或更高版本，Comctl32.dll 就会更换为新版本，其中包含现有控件的升级版本和几个新公用控件类型。新控件类型之一是扩展后的组合框控件，常称为 ComboBoxEx 控件。ComboBoxEx 控件和标准组合框控件在几个重要方面都不相同。最值得注意的是它能在每个列表项旁边显示图像。自然，这些图像来自于图像列表。可以将图像列表和 ComboBoxEx 控件合起来用，创建一个既包含图形又包含文本的下拉列表。

下面粗略介绍一下图像列表和 ComboBoxEx 控件，同时谈谈应用中要注意的几个基本原则。然后，通过开发一个以可视化方式描述路径名的组合框，您就会了解图像列表和 ComboBoxEx控件的强大功能了。

16.3.1　图像列表

图像列表是由一系列尺寸相同的位图图像组成并形成一逻辑单位的集合。MFC 的

CImageList 类提供了许多函数,如创建图像列表的函数,增加和删除图像的函数,在屏幕上显示图像的函数,保存和读取图像列表的函数等等。图像列表非常有用,因为在它上面实现的许多功能在 Windows 中都是特有的。但是首先要把图像列表添加到操作系统中,这样,位图才能组合成一个单位并传送给其他公用控件。例如:如果要给树形视图控件提供图像,请不要传递给它 CBitmaps 数组;而应传递给它图像列表的句柄(HIMAGELIST)或指向 CImageList 对象的指针。

图像列表的样子可以想像成图像沿水平方向从一端排到另一端的电影胶片。位于左端的为图像 0,右侧相邻的则为图像 1,如此类推。图像的高和宽是任意的,但是所有图像的高和宽必须是一致的。

MFC 提供三种方式创建图像列表:先创建空的图像列表,然后用 CImageList::Add 在其中添加图像;用包含图像数组的位图创建初始化了的图像列表;还可以通过合并现有图像列表的图像创建初始化了的图像列表。重载后的 CImageList::Create 支持三种创建方法。第二种方法(可能是最常用的)用下面的例子说明。假定 IDB_BITMAP 是位图的资源 ID。该位图包含五个图像,每个图像有 18 个像素宽、16 个像素高。位图自身是 90 个像素宽(18 的 5 倍)、16 个像素高。下列语句基于这个位图创建了一个图像列表:

```
CImageList il;
il.Create (IDB_BITMAP, 18, 1, CLR_NONE);
```

传给 Create 的第一个参数是位图的资源 ID,也可以向这个参数传递字符串资源 ID。第二个参数是单个图像的宽,以像素表示。Windows 通过用图像宽划分位图宽确定列表中添加图像的个数。第三个参数是"增长尺寸"。图像列表的大小和由 MFC 基本类创建的数组一样可以动态调整;当添加新图像时,增长尺寸会告诉图像列表,需要为多少个附加图像分配内存。最后一个参数(CLR_NONE)创建了一个"非屏蔽的"图像列表。非屏蔽的图像是一些普通位图,在屏幕上显示这些位图时,各位块是直接传送到目的地的。

如果给 CImageList::Create 传递的参数值为 COLORREF(而不是 CLR_NONE),则创建的是"屏蔽的"图像列表。除了要保存屏蔽图像的颜色信息,Windows 还保存单色位屏蔽,使它能区分前景像素和背景像素。传递给 CImageList::Create 的 COLORREF 值指定了背景色,任何设置成背景色的像素都被认为是背景像素。屏蔽图像的优点是:在画图之前,可以在图像列表中调用 CImageList::SetBkColor 把背景色设置成自己喜欢的颜色。原位图中的背景色可能是品红,但是如果您将背景色设置成红色并画出图像,则所有品红色的像素都会变成红的。最妙的是:如果您传递给 CImageList::SetBkColor 一个 CLR_NONE 参数,背景像素就根本不会画出来。因此,图像列表提供了一种简单方法来绘制具有透明像素的位图。还记得第 15 章画非矩形位图的 DrawTransparent 函数吗?图像列表可以使您用更少的代码完成这个工作,而且图像列表方法更快,因为图像分块向屏幕传送时不必每次重画屏蔽面。

CImageList::Draw 在屏幕上显示图像。下一语句将列表中第 3 个图像(图像 2)显示在屏

幕 DC（由 CDC 指针 pDC 引用）上：

```
il.Draw(pDC, 2, point, ILD_NORMAL);
```

point 是一个 POINT 结构，包含图像左上角在目标 DC 中的 x 和 y 坐标值。ILD_NORMAL 是一个标志，告诉 Draw 函数用当前背景色画屏蔽图像。（该标志对非屏蔽图像不起作用。）如果不论当前背景色是什么，都希望背景像素是透明的，则可以改用 ILD_TRANSPARENT 标志：

```
il.Draw(pDC, 2, point, ILD_TRANSPARENT);
```

如果要获得一些有趣的效果，可以试着用 ILD_BLEND25 或 ILD_BLEND50 标志显示屏蔽图像，使它和系统加亮色（COLOR_HIGHLIGHT）混合起来。CImageList::Draw 还接受 ILD_SELECTED和 ILD_FOCUS 标志，但实际上它们和 ILD_BLEND50 和 ILD_BLEND25 没有多大区别。如果想看到色彩混合的效果，可在 Windows 桌面上选择一个图标。如果要显示处于选中状态的图标，系统通过以 ILD_BLEND50 标志画图标，使图标成为亮色。

此外，用 ILD_TRANSPARENT 标志或以设置为 CLR_NONE 的背景色画图像总是要比画非屏蔽图像慢一点。如果要把包含透明像素的图像显示到单色背景中，则用 CImageList::SetBkColor 将图像列表的背景色设置成单色背景的颜色，然后以 ILD_NORMAL 标志调用 CImageList::Draw。这样既提高了程序性能，又得到了想要的透明像素。

16.3.2　ComboBoxEx 控件

ComboBoxEx 控件可以简化向组合框中添加图像和文本的工作。假定 m_wndCBEx 是对话框类的一个 ComboBoxEx 数据成员，m_wndCBEx 被映射成对话框中的 ComboBoxEx 控件，m_il 是 CImageList 的实例，则下列语句用标记为“Item 1”到“Item 5”的五项初始化控件。每项旁边都有一个从图像列表提取出的文件夹图像，而图像列表又是从位图资源 IDB_IMAGE获得的图像：

```
m_il.Create(IDB_IMAGE, 16, 1, RGB(255, 0, 255));
m_wndCBEx.SetImageList(&m_il);

for(int i = 0; i < 5; i++)
  CString string;
  string.Format(_T("Item %d"), i);

  COMBOBOXEXITEM cbei;
  cbei.mask = CBEIF_IMAGE|CBEIF_SELECTEDIMAGE|CBEIF_TEXT;
  cbei.iItem = i;
  cbei.pszText = (LPTSTR)(LPCTSTR)string;
  cbei.iImage = 0;
  cbei.iSelectedImage = 0;
```

```
    m_wndCBEx.InsertItem (&cbei);
}
```

该示例使用的主要函数包括：CComboBoxEx::SetImageList，它将图像列表和 ComboBoxEx 控件关联起来；CComboBoxEx::InsertItem，它在控件中添加一项。InsertItem 接受指向 COMBOBOXEXITEM 结构的指针。该结构包含添加项的关键信息，其中包括该项的文本和关联图像的基于 0 的索引值(如果有的话)。该项未选中时，iImage 指定显示在该项旁的图像；该项选中时，iSelectedImage 指定已显示的图像。图 16-7 给出最终生成的控件，并显示了控件的下拉列表。

图 16-7 包含文字和图像的 ComboBoxEx 控件

通过在 COMBOBOXEXITEM 的 iIndent 域指定“空格”数，可以在 ComboBoxEx 控件中缩进一项。每个空格等于 10 个像素。下面这个示例将一个 ComboBoxEx 控件初始化。除了控件中每个连续项要比前个示例中的多缩进一个空格外，该控件和前个示例中的 ComboBoxEx 控件完全相同。

```
m_il.Create (IDB_IMAGE, 16, 1, RGB (255, 0, 255));
m_wndCBEx.SetImageList (&m_il);

for (int i = 0; i < 5; i ++) {
  CString string;
  string.Format (_T ("Item %d"), i);
  COMBOBOXEXITEM cbei;
  cbei.mask = CBEIF_IMAGE | CBEIF_SELECTEDIMAGE | CBEIF_TEXT|
    CBEIF_INDENT;
  cbei.iItem = i;
  cbei.pszText = (LPTSTR) (LPCTSTR) string;
  cbei.iImage = 0;
  cbei.iSelectedImage = 0;
  cbei.iIndent = i;
  m_wndCBEx.InsertItem (&cbei);
}
```

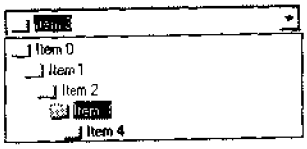

图 16-8 包含缩进项的 ComboBoxEx 控件

结果显示在图 16-8 中。对要用 ComboBoxEx 控件显示彼此间有等级关系的项目目，比如组成路径名的多个目录名时，能把项目缩进任意个空格的这一功能提供了不少。CComboBoxEx 有四个控制 ComboBoxEx 控件中各项的成员函数，InsertItem是其中之一。其余三个是：用于清除某项的 DeleteItem，将某项信息复制到 COMBOBOXITEM 结构中的 GetItem，以及根据 COMBOBOXITEM 结构提供的信息修改某项的 SetItem。

CComboBoxEx 有自己的一套成员函数。常见操作，如选中某项或获取选中项的索引值，都是用 CComboBox 成员函数实现的。因为 CComboBoxEx 是从 CComboBox 派生来的，所以可以对 CComboBoxEx 调用 CComboBox 函数。例如：语句

```
m_wndCBEx.SetCurSel (nIndex);
```

选中索引值为 nIndex 的一项，而语句

```
int nIndex = m_wndCBEx.GetCurSel ();
```

将 nIndex 设成当前选中项的索引值。和传统组合框相同，ComboBoxEx 控件共分三类：简单式、下拉式和下拉列表式，各自对应着 CBS_SIMPLE、CBS_DROPDOWN 或 CBS_DROPDOWNLIST。其他 CBS 样式，如 CBS_SORT，不能用于 ComboBoxEx 控件；即使使用了它们，结果也必然是被忽略的。然而，ComboBoxEx 控件还支持自己的几种样式，常称为“扩展样式”。它们不能用在对话框模板或 Create 语句中。必须在控件创建后用 CComboBoxEx::SetExtendedStyle 编程实现这些样式的应用。表 16-9 给出了几种扩展样式。所有平台都支持这几种样式。例如：为了使控件中的文本是上下文敏感的，可以这样实现：

```
m_wndCBEx.SetExtendedStyle (CBES_EX_CASESENSITIVE,
  CBES_EX_CASESENSITIVE);
```

SetExtendedStyle 的第 2 个参数指定了需要的一个样式或多个样式。第一个参数是样式屏蔽，有了样式屏蔽，其他样式就不会受到影响。如果参数 1 为零，则屏蔽被清除。

表 16-9　COMBOBOXEX 控件的扩展样式

样式	说　明
CBES_EX_CASESENSITIVE	使字符串搜索具有上下文敏感性
CBES_EX_NOEDITIMAGE	禁止各项获得图像
CBES_EX_NOEDITIMAGEINDENT	禁止各项获得图像和使用左缩进，以取消留给图像用的空间
CBES_EX_NOSIZELIMIT	允许 ComboBoxEx 控件的高度比被包含在里面的组合框的高度小

ComboBoxEx 控件发送给父窗口的 CBN 通知和传统组合框发送的一样。它还支持自己特有的通知，请见表 16-10。

表 16-10　ComboBoxEx 通知

通知	发送时间
CBEN_BEGINEDIT	用户显示控件的下拉列表，或单击编辑控件开始编辑
CBEN_ENDEDIT	用户在控件列表框中选择，或直接编辑控件的文本
CBEN_DRAGBEGIN	用户在控件中拖动某项，执行拖放操作

续表

通知	发送时间
CBEN_INSERTITEM	某项被添加到控件中
CBEN_DELETEITEM	控件中某项被清除
CBEN_GETDISPINFO	在显示某项前,控件需要该项的附加信息,如文本字符串、图像、缩进或一些的组合信息
NM_SETCURSOR	响应 WM_SETCURSOR 消息,控件准备设置光标

MFC 应用程序通过 ON_NOTIFY 宏将 CBEN 通知映射为父窗口类中的处理函数。只有在(传递给 InsertItem 的)COMBOBOXEXITEM 结构的 pszText 域包含 LPSTR_TEXTCALLBACK, iImage 或 iSelectedImage 域包含 I_IMAGECALLBACK, 或者 iIndent 域包含 I_INDENTCALLBACK 时,才发送 CBEN_GETDISPINFO 通知。可以用这些特殊值创建动态 ComboBoxEx 控件。该控件在任意时刻都具有文本、图像和缩进。

16.3.3 PathList 应用程序

PathList(如图 16-9 所示)是一个基于对话框的 MFC 应用程序。它使用了 ComboBoxEx 控件描述路径名。控件是 CPathComboBox 的一个实例。CPathComboBox 是从 CComboBoxEx 派生来的,它具有两个公用成员函数: SetPath 和 GetPath。得到一个正确路径名时,SetPath 分析路径名,并添加代表各目录的缩进项。(SetPath 检查驱动器字母,但不检查路径名的其他部分。)GetPath 返回当前控件中由选中驱动器和目录组成的正确路径名。

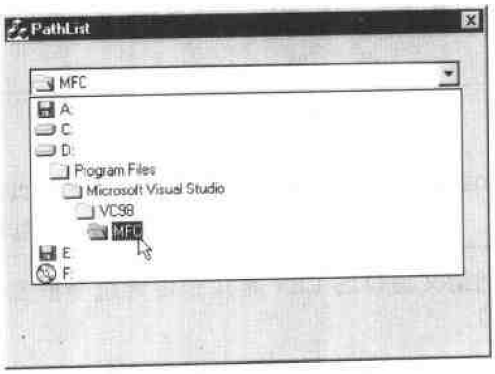

图 16-9 PathList 窗口

PathList 对话框类和 ComboBoxEx 类的源代码在图 16-10 中。PathList 的对话框窗口只是添加了 ComboBoxEx 控件,基本上再没对控件做其他工作。运行时,它通过当前路径名调用 SetPath;在选中某项时,它显示由 GetPath 返回的路径名。控件类 CPathComboBox 包含许多有趣的成分,包括分析(传递给 SetPath 的)路径名的代码;路径名更换时,清除原有项的代码,

等等。如果花点时间了解该程序是如何工作的，您就容易理解 ComboBoxEx 控件了。

PathListDlg.h

```
// PathListDlg.h : header file
//

#if !defined(
  AFX_PATHLISTDLG_H__710413E6_AC66_11D2_8E53_006008A82731__INCLUDED_)
#  define  AFX_PATHLISTDLG_H__710413E6_AC66_11D2_8E53_006008A82731__INCLUD-
ED_

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

/////////////////////////////////////////////////////////////////////////////
// CPathListDlg dialog

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

// Dialog Data
  //{{AFX_DATA(CPathListDlg)
  enum { IDD = IDD_PATHLIST_DIALOG };
  CPathComboBox  m_wndCBEx;
  //}}AFX_DATA

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

// Implementation
protected:
  HICON m_hIcon;

  // Generated message map functions
  //{{AFX_MSG(CPathListDlg)
  virtual BOOL OnInitDialog();
  afx_msg void OnPaint();
```

```
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnSelEndOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

#endif
// !defined(
//   AFX_PATHLISTDLG_H__710413E6_AC66_11D2_8E53_006008A82731__INCLUDED_)
```

PathListDlg.cpp

```
// PathListDlg.cpp : implementation file
//

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

/////////////////////////////////////////////////////////////////////////////
// CPathListDlg dialog

CPathListDlg::CPathListDlg(CWnd* pParent /* = NULL */)
    : CDialog(CPathListDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPathListDlg)
    //}}AFX_DATA_INIT
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPathListDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPathListDlg)
    DDX_Control(pDX, IDC_CBEX, m_wndCBEx);
    //}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CPathListDlg, CDialog)
  //{{AFX_MSG_MAP(CPathListDlg)
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  ON_CBN_SELENDOK(IDC_CBEX, OnSelEndOK)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPathListDlg message handlers

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

  SetIcon(m_hIcon, TRUE);
  SetIcon(m_hIcon, FALSE);

  //
  // Initialize the ComboBoxEx control.

  //  TCHAR szPath[MAX_PATH];
  ::GetCurrentDirectory(sizeof(szPath) / sizeof(TCHAR), szPath);
  m_wndCBEx.SetPath(szPath);
  return TRUE;
}

void CPathListDlg::OnPaint()
{
  if (IsIconic())
  {
    CPaintDC dc(this); // device context for painting

    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
```

```
    dc.DrawIcon(x, y, m_hIcon);
  }
  else
  {
    CDialog::OnPaint();
  }
}

HCURSOR CPathListDlg::OnQueryDragIcon()
{
  return (HCURSOR) m_hIcon;
}

void CPathListDlg::OnSelEndOK()
{
  //
  // Display the path just selected from the ComboBoxEx control.
  //
  MessageBox (m_wndCBEx.GetPath ());
}
```

PathComboBox.h

```
#if !defined(
  AFX_PATHCOMBOBOX_H__710413F1_AC66_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_PATHCOMBOBOX_H__710413F1_AC66_11D2_8E53_006008A82731__INCLUD-
ED_

#if_MSC_VER > 1000
#pragma once
#endif //_MSC_VER > 1000
// PathComboBox.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPathComboBox window

class CPathComboBox : public CComboBoxEx
{
// Construction
public:
  CPathComboBox();

// Attributes
public:
```

```
// Operations
public:

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

// Implementation
public:
  CString GetPath();
  BOOL SetPath (LPCTSTR pszPath);
  virtual ~CPathComboBox();

  // Generated message map functions
protected:
  void GetSubstring (int& nStart, CString& string, CString& result);
  int m_nIndexEnd;
  int m_nIndexStart;
  BOOL m_bFirstCall;
  CImageList m_il;
  //{{AFX_MSG(CPathComboBox)

  //}}AFX_MSG

  DECLARE_MESSAGE_MAP()
};

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

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

#endif
// !defined(
//   AFX_PATHCOMBOBOX_H__710413F1_AC66_11D2_8E53_006008A82731__INCLUDED_)
```

PathComboBox.cpp

```
// PathComboBox.cpp : implementation file
//

#include "stdafx.h"
#include "PathList.h"
#include "PathComboBox.h"
```

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

/////////////////////////////////////////////////////////////////////////////
// CPathComboBox

CPathComboBox::CPathComboBox()
{
  m_bFirstCall = TRUE;
  m_nIndexStart = -1;
  m_nIndexEnd = -1;
}

CPathComboBox::~CPathComboBox()
{
}

BEGIN_MESSAGE_MAP(CPathComboBox, CComboBoxEx)
  //{{AFX_MSG_MAP(CPathComboBox)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPathComboBox message handlers

BOOL CPathComboBox::SetPath(LPCTSTR pszPath)
{
  if (m_bFirstCall) {
    m_bFirstCall = FALSE;

    //
    // Add an image list containing drive and folder images.
    //
    m_il.Create (IDB_IMAGES, 16, 1, RGB (255, 0, 255));
    SetImageList (&m_il);

    //
    // Add icons representing the drives on the host system.
    //
    int nPos = 0;
    int nCount = 0;
    CString string = _T ("?:\\");
```

```
    DWORD dwDriveList = ::GetLogicalDrives();

    while (dwDriveList) {
      if (dwDriveList & 1) {
        string.SetAt(0,_T('A') + nPos);
        CString strDrive = string.Left(2);
        UINT nType = ::GetDriveType(string);

        int nImage = 0;
        switch (nType) {

        case DRIVE_FIXED:
          nImage = 0;
          break;

        case DRIVE_REMOVABLE:
          nImage = 1;
          break;

        case DRIVE_CDROM:
          nImage = 2;
          break;

        case DRIVE_REMOTE:
          nImage = 3;
          break;
        }

        COMBOBOXEXITEM cbei;
        cbei.mask = CBEIF_TEXT | CBEIF_IMAGE | CBEIF_SELECTEDIMAGE;
        cbei.iItem = nCount++;
        cbei.pszText = (LPTSTR)(LPCTSTR) strDrive;
        cbei.iImage = nImage;
        cbei.iSelectedImage = nImage;
        InsertItem(&cbei);
      }
      dwDriveList >>= 1;
      nPos++;
    }
  }

  //
  // Find the item that corresponds to the drive specifier in pszPath.
  //
  CString strPath = pszPath;
  CString strDrive = strPath.Left(2);
```

```
int nDriveIndex = FindStringExact ( -1, strDrive);
if (nDriveIndex == CB_ERR)
  return FALSE;

//
// Delete previously added folder items (if any).
//
if (m_nIndexStart != -1 && m_nIndexEnd != -1) {
  ASSERT (m_nIndexEnd >= m_nIndexStart);
  int nCount = m_nIndexEnd - m_nIndexStart + 1;
  for (int i = 0; i < nCount; i++)
    DeleteItem (m_nIndexStart);
  if (m_nIndexStart < nDriveIndex)
    nDriveIndex -= nCount;
  m_nIndexStart = -1;
  m_nIndexEnd = -1;
}

//
// Add items representing the directories in pszPath.
//
int nCount = 0;
int nStringIndex = strPath.Find (_T ('\\'), 0);

if (nStringIndex++ != -1) {
  CString strItem;
  GetSubstring (nStringIndex, strPath, strItem);

  while (!strItem.IsEmpty ()) {
    COMBOBOXEXITEM cbei;
    cbei.mask = CBEIF_TEXT | CBEIF_IMAGE | CBEIF_SELECTEDIMAGE |
      CBEIF_INDENT;
    cbei.iItem = nDriveIndex +++nCount;
    cbei.pszText = (LPTSTR) (LPCTSTR) strItem;
    cbei.iImage = 4;
    cbei.iSelectedImage = 5;
    cbei.iIndent = nCount;
    InsertItem (&cbei);

    GetSubstring (nStringIndex, strPath, strItem);
  }
}

//
// Record the indexes of the items that were added, too.
```

```
    //
    if (nCount) {
      m_nIndexStart = nDriveIndex + 1;
      m_nIndexEnd = nDriveIndex + nCount;
    }

    //
    // Finish up by selecting the final item.
    //
    int nResult = SetCurSel (nDriveIndex + nCount);
    return TRUE;
}

void CPathComboBox::GetSubstring(int& nStart, CString &string,
  CString &result)
{
  result = _T ("");
  int nLen = string.GetLength ();

  if (nStart >= nLen)
    return;

  int nEnd = string.Find (_T ('\\'), nStart);
  if (nEnd == -1) {
    result = string.Right (nLen - nStart);
    nStart = nLen;
  }
  else {
    result = string.Mid (nStart, nEnd - nStart);
    nStart = nEnd + 1;
  }
}

CString CPathComboBox::GetPath()
{
  //
  // Get the index of the selected item.
  //
  CString strResult;
  int nEnd = GetCurSel ();
  int nStart = nEnd + 1;

  //
  // Find the index of the "root" item.
  //
  COMBOBOXEXITEM cbei;
```

```
    do {
      cbei.mask = CBEIF_INDENT;
      cbei.iItem = --nStart;
      GetItem(&cbei);
    } while (cbei.iIndent != 0);

    //
    // Build a path name by combining all the items from the root item to
    // the selected item.
    //
    for (int i = nStart; i <= nEnd; i++) {
      TCHAR szItem[MAX_PATH];
      COMBOBOXEXITEM cbei;
      cbei.mask = CBEIF_TEXT;
      cbei.iItem = i;
      cbei.pszText = szItem;
      cbei.cchTextMax = sizeof(szItem) / sizeof(TCHAR);
      GetItem(&cbei);

      strResult += szItem;
      strResult += _T("\\");
    }

    //
    // Strip the trailing backslash.
    //
    int nLen = strResult.GetLength();
    strResult = strResult.Left(nLen - 1);
    return strResult;
```

图 16-10 PathList 应用程序

16.4 进度控件和动画控件

在向用户提供任务状态反馈信息方面，Comctl32.dll 还有两种好用的用于反馈用户所关心的任务状态信息的工具。第一个是进度控件。“进度控件”是一个竖直或水平放置的矩形，它包含一个颜色棒，随着任务的不断执行，这个棒也在增长。第二个是“动画控件”，它把播放 AVI 文件的复杂工作简化为两个函数调用。尽管动画控件有多种用途，但它们常常只用来让用户了解大型任务的进行情况。

16.4.1　进度控件

MFC 用 CProgressCtrl 类的实例表示进度控件。在默认方式下，进度控件沿水平方向放置，棒体用一系列线段表达。通过赋给进度控件样式 PBS_VERTICAL，可以把控件竖直放置，并且还可以用 PBS_SMOOTH 将虚线改为实线（如图 16-11 所示）　不过，两种样式都要求有 Internet Explorer 3.0 或更高版本的支持。

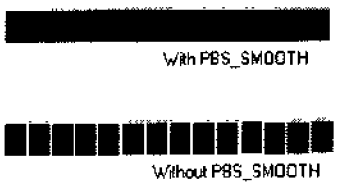

图 16-11　具有和不具有样式 PBS_SMOOTH 的进度控件

和滑杆控件相同，进度控件也有范围和位置。如果进度控件的范围是 0 到 100、位置是 20，则棒体填充了控件的 20%。如果范围是 100 到 400，位置是 300，则棒体为控件长度的 2/3。默认范围是 0 到 100，但是利用 CProgressCtrl::SetRange，可以把它改成任意值。如果 m_ wndProgress是一个 CProgressCtrl 对象，语句

```
m_wndProgress.SetRange(100, 400);
```

将控件范围设成 100 到 400　SetRange 将最小值和最大值限制在 16 位值，但是如果系统安装了 Internet Explorer 3.0 或更高版本，通过 CProgressCtrl::SetRange32 这个新函数，您可以指定 32 位的范围大小。如果要获取当前范围，可以使用 GetRange。该函数对 16 位和 32 位范围都有效

在创建进度控件并设定范围之后，可以用 CProgressCtrl::SetPos 设定位置　下面这个示例通过在循环中反复调用 SetPos 使进度控件从 0 到 100 每隔 2½秒前进一步。其中，为了确保每次迭代至少占用 25 毫秒，该循环使用了 ::Sleep API 函数：

```
m_wndProgress.SetRange(0, 100);
m_wndProgress.SetPos(0);
for (int i=0; i<100; i++) {
  m_wndProgress.SetPos(i);
  ::Sleep(25);
}
m_wndProgress.SetPos(0);
```

这是使进度控件分步前进的一种方法。还可以用 OffsetPos 函数指定一个相对于当前位置的新位置。使用 OffsetPos 重写上段代码:

```
m_wndProgress.SetRange (0, 100);
m_wndProgress.SetPos (0);
for (int i = 0; i < 100; i ++ ) {
  m_wndProgress.OffsetPos (1);
  ::Sleep (25);
}
m_wndProgress.SetPos (0);
```

第 3 种方法是:用 SetStep 赋给控件一个步长,然后通过 StepIt 根据当前步长增加位置。

```
m_wndProgress.SetRange (0, 100);
m_wndProgress.SetPos (0);
m_wndProgress.SetStep (1);
for (int i = 0; i < 100; i ++ ) {
  m_wndProgress.StepIt ();
  ::Sleep (25);
}
m_wndProgress.SetPos (0);
```

可以在任意时刻调用 CProgressCtrl::GetPos 函数,获取控件的当前位置。

在默认方式下,进度控件中棒体的颜色是系统颜色 COLOR_HIGHLIGHT,控件的背景颜色是 COLOR_3DFACE。在装有 Internet Explorer 4.0 或更高版本的系统上,可以通过 PBM_SETBARCOLOR 消息改变棒体颜色,通过 PBM_SETBKCOLOR 消息改变控件的背景颜色。因为 CProgressCtrl 缺乏这些消息的封装函数,所以您必须自己发送消息。例如:语句

```
m_wndProgress.SendMessage (PBM_SETBARCOLOR, 0, (LPARAM) RGB (255, 0, 0));
```

将 m_wndProgress 棒体颜色改成红色。

第 17 章的一个示例程序 ImageEdit 使用了进度控件显示图像处理任务的进行情况。进度控件附着在状态栏上。在任务执行前,进度控件像是一个普通的状态栏窗格;执行时,棒体出现,并不断在控件表面增长。如果您想看到动作中的进度控件,可以跳过本章,看看 ImageEdit。

16.4.2 动画控件

动画控件简化了在对话框或窗口中播放影像剪辑的工作。剪辑必须是 Windows AVI 格式,而且其中最多可以有两个流。如果其中一个是音频流,则被忽略。Visual Studio 带有许多 AVI 文件,用它们作动画控件效果很好。其中一个示例文件,Findfile.avi,包含绕圈运动

的放大镜，用在系统的 Find 公用程序中。另一个，Filecopy.avi，包含“飞行中的纸张”剪辑。在您把大型文件或一组文件从一个文件夹拖放或拷贝到另一个文件夹中时，这个动画过程就会出现。

CAnimateCtrl 将动画控件的功能封装在一个方便易用的 C++ 类中。使用 CAnimateCtrl 本身很简单。CAnimateCtrl::Open 从资源文件或外部文件中加载 AVI 剪辑。CAnimateCtrl::Play 则开始播放剪辑，CAnimateCtrl::Stop 停止播放，CAnimateCtrl::Close 卸载剪辑。假定 m_wndAnimate是 CAnimateCtrl 的一个实例，并且关联着一个动画控件。下列程序代码加载 AVI 文件 Findfile.avi，并开始播放它：

```
m_wndAnimate.Open (_T ("Findfile.avi"));
m_wndAnimate.Play (0, -1, -1);
```

Open 不接收文件名而接收资源 ID，可以让您把 AVI 剪辑作为资源嵌在 EXE 文件中：

```
// In the RC file
IDR_FINDFILE AVI "Findfile.avi"
  .
  .
  .
// In the CPP file
m_wndAnimate.Open (IDR_FINDFILE);
m_wndAnimate.Play (0, -1, -1);
```

Play 启动动画过程并立刻返回，而不是动画播放完后才返回。这样的好处在于动画在背景中播放时调用 Play 的线程还可以继续工作。

Play 接收三个参数：起始和终止帧号，指定的动画播放次数。如果把三个参数分别指定为 0、-1 和 1，则播放所有帧，并反复播放直到 Stop 被调用：

```
m_wndAnimate.Stop ();
```

调用 Stop 之后，如果不打算再播放它，则应该调用 Close 清除内存中的剪辑：

```
m_wndAnimate.Close ();
```

Open 调用和 Close 调用应该成对出现，以免耗费资源。

动画控件支持四种样式。这些样式会影响控件的外观和动作执行。ACS_AUTOPLAY 设置控件，使它一经打开，不必等到 Play 被调用就开始播放动画。ACS_CENTER 使输出位于控件矩形的中间。如果没有这个样式，剪辑就会播放在控件矩形的左上角，控件就会重新调整大小直到和动画中的帧大小相等。ACS_TRANSPARENT 用透明背景播放动画，而不用 AVI 文件中指定的背景色。最后，ACS_TIMER 禁止控件执行后台线程，实现自己的画图任务。ACS_TIMER 样式的动画控件并没有启动其他线程(线程消耗资源，太多的线程会使系统瘫痪)，而是在调用者的线程中设置了一个计时器，并利用计时器的回调函数画出连续的

帧。只有安装了 Internet Explorer 3.0 或更高版本的系统才支持 ACS_TIMER。

16.5 IP 地址控件和其他数据输入控件

IP 地址控件、热键控件、月历控件和日期-时间拾取控件都有一个共同的特点：它们能使格式化输入工作变得简单。其中有一些，如 IP 地址控件，是极其简单的；其他的，如日期-时间拾取控件，提供了极多的选项。然而，所有这些控件都非常容易编程实现，如果要使用 MFC 提供的封装类，则更加容易。下面就概括地介绍一下这四种控件类型，并给出程序示例示范它们的用法。

16.5.1 IP 地址控件

IP 地址控件简化了 32 位 IP 地址(它包括四个由句点分隔的 8 位整型值，如 10.255.10.1)的输入。控件只接受数字，并被分成四个可以放 3 位数的域，如图 16-12 所示。用户在域中敲入三位数时，输入焦点自动移到下一个域。只有在安装了 Internet Explorer 4.0 或更高版本的系统中才有 IP 地址控件。

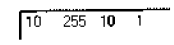

图 16-12 IP 地址控件

MFC 通过 CIPAddressCtrl 将 IP 地址控件的接口代码化。名为 SetAddress 和 GetAddress 的 CIPAddressCtrl 函数输入或输出 IP 地址。如果 m_wndIPAddress 是对话框类的一个 CIPAddressCtrl 数据成员，下面的 OnInitDialog 和 OnOK 函数在创建对话框时用保存在 m_nField1 到 m_nField4 中的 IP 地址初始化控件；在清除对话框时，从控件中提取 IP 地址：

```
// In CMyDialog's class declaration
BYTE m_nField1, m_nField2, m_nField3, m_nField4
  .
  .
  .
BOOL CMyDialog::OnInitDialog ()
{
  CDialog::OnInitDialog ();
  m_wndIPAddress.SetAddress (m_nField1, m_nField2,
    m_nField3, m_nField4);
  return TRUE;
}

void CMyDialog::OnOK ()
```

```
{
  m_wndIPAddress.GetAddress (m_nField1, m_nField2,
    m_nField3, m_nField4);
  CDialog::OnOK ();
}
```

也可以用 CIPAddressCtrl::ClearAddress 清空 IP 地址控件，或者用 CIPAddressCtrl::IsBlank 确定控件是否为空。可以用另一个 CIPAddressCtrl 成员函数 SetFieldFocus 将输入焦点移到指定区域。

在默认方式下，IP 地址控件中的每个域接受 0 到 255 间的值。可以通过 CIPAddressCtrl::SetFieldRange 改变某个指定域的接受范围。下面这个语句设置控件，使输入控件第一个域的值限制在 10 到 100 之间，最后一个域的值限制在 100 到 155 之间：

```
m_wndIPAddress.SetFieldRange (0, 10, 100); // Field 1
m_wndIPAddress.SetFieldRange (3, 100, 155); // Field 4
```

控件禁止非法值输入某域，它会自动把范围外的值适当地转换成范围的上限或下限。

IP 地址控件向它的所有者发送四种通知。EN_SETFOCUS 和 EN_KILLFOCUS 通知表示控件得到或失去输入焦点；EN_CHANGE 通知表示控件中的数据改变了。三种通知都封装在 WM_COMMAND 消息中。域值发生变化或输入焦点移至另一个域中时，IP 地址控件还发送 IPN_FIELDCHANGED 通知。IPN_FIELDCHANGED 在所有 IP 地址控件通知中是独特的，它由 WM_NOTIFY 消息携带着传送。

16.5.2　热键控件

热键控件和 IP 地址控件在概念上是类似的。主要的区别在于：热键接受键的组合，而不是 IP 地址。在本质上热键控件是个很出色的编辑控件，它能自动将键的组合，如 Ctrl-Alt-P，转换为适合在屏幕上显示的文本字符串。热键控件之所以得名，是因为输入的键组合通过 WM_SETHOTKEY 消息有时会被转换成热键。然而，输入热键控件的数据不一定要用于热键，开发者可以按照自己认为合适的方式用它。

MFC 用 CHotKeyCtrl 的实例代表热键控件。成员函数 SetHotKey 和 GetHotKey 把键组合转换为控件显示的文本字符串，或者反过来把文本字符串转换为键组合。下面这个语句用键组合 Ctrl-Alt-P 初始化由 CHotKeyCtrl 对象 m_wndHotkey 代表的热键控件。控件响应，显示文本字符串“Ctrl + Alt + P”：

```
m_wndHotkey.SetHotKey ('P', HOTKEYF_CONTROL | HOTKEYF_ALT);
```

下面两个语句将热键控件中的数据读入变量 wKeyCode（它保存有虚拟键代码）和 wModifiers（它保存有位标志，确定键组合中所包含（如果有的话）的控制键）：

```
WORD wKeyCode, wModifiers;
m_wndHotkey.GetHotKey (wKeyCode, wModifiers);
```

还可以在对话框类的 OnInitDialog 和 OnOK 函数中包含 SetHotKey 和 GetHotKey 之类的调用，实现热键控件和对话框类数据成员间的数据传输。

默认方式下，热键控件接受包含 Ctrl、Shift 和 Alt 键的任意组合的键组合。通过调用 CHotKeyCtrl::SetRules 可以限定控件能接受的组合。SetRules 接收两个参数：确定 Ctrl、Shift 和 Alt 为无效组合的位标志数组，以及指定作为替代的合法组合的位标志数组。例如：语句

```
m_wndHotkey.SetRules (HKCOMB_A | HKCOMB_CA | HKCOMB_SA | HKCOMB_SCA, 0);
```

禁止包含 Alt 键的键组合，而语句

```
m_wndHotkey.SetRules (HKCOMB_A | HKCOMB_CA | HKCOMB_SA | HKCOMB_SCA,
  HOTKEYF_CONTROL);
```

也能达到这个目的，同时还指示控件用 Ctrl 键替代包含 Alt 键的键组合中的所有控制键。如果想了解其他可支持的 HKCOMB 标志，请参见 SetRules 说明文档。

16.5.3 月历控件

月历控件(也叫日历控件)可以让用户在日历中拾取数据，从而输入日期，而不必把它们敲入编辑控件（参见图 16-13）。日历控件可以支持单项或多项选择。在单选日历控件中单击一个日期会把该日期设成“当前日期”。在多选日历控件中，用户可以选择单个日期或一个日期段。无论是单个日期还是一个日期段，您都可以通过编程向控件发送消息设置或提取当前选项。MFC 将所有日历控件消息封装在 CMonthCalCtrl 类的成员函数中。

图 16-13 月历控件

在单选日历控件中，CMonthCalCtrl::SetCurSel 设置当前日期；而 CMonthCalCtrl::GetCurSel 提取日期。语句

```
m_wndCal.SetCurSel (CTime (1999, 9, 30, 0, 0, 0));
```

将当前日期设成 1999 年 9 月 30 日，该日期在日历控件中由 m_wndCal 代表。表面上，语句

```
CTime date;
m_wndCal.GetCurSel (date);
```

在用当前所选日期初始化的“日期”控件中提取日期。但要小心，和说明文档中讲的相反，日历控件有时会返回 SYSTEMTIME 结构的时、分、秒和毫秒域中的随机数。为响应

MCM_GETCURSEL消息，控件要用该结构显示日期。因为 CTime 在处理从 SYSTEMTIME 结构获取的时间时也考虑了日期的计算特点，例如：如果是 25 小时则加一天，所以由 CMonthCalCtrl::GetCurSel 初始化的 CTime 对象不可信赖。解决方法是：首先给控件发送 MCM_GETCURSEL消息，然后在 SYSTEMTIME 结构的时间域转换为 CTime 之前把时间域置零，最终得到当前日期，如下所示：

```
SYSTEMTIME st;
m_wndCal.SendMessage (MCM_GETCURSEL, 0, (LPARAM) &st);
st.wHour = st.wMinute = st.wSecond = st.wMilliseconds = 0;
CTime date (st);
```

如果您喜欢，还可以用 CMonthCalCtrl 的 SetRange 函数设置允许用户选择的日期上、下界。

可以代替 SetCurSel 和 GetCurSel 输入和输出数据的是 DDX。MFC 含有一个 DDX 函数 DDX_MonthCalCtrl，您可以在对话框的 DoDataExchange 函数中调用该函数，实现月历控件和 CTime 或 COleDateTime 数据成员间的数据自动传输。MFC 甚至还包含用于数据合法性检查的 DDV 函数。但是猜猜出现什么问题了？因为 DDX_MonthCalCtrl 使用 GetCurSel 读取当前日期，所以实际上它不起什么作用。在这个错误解决之前，您最好还是放弃 DDX，而使用前面介绍的方法输入和输出当前日期。

通过在控件样式中设置 MCS_MULTISELECT 位，可以创建一个日历控件，使它允许用户选取一个连续的日期范围。在默认方式下，选中范围不能超过 7 天。可以用 CMonthCalCtrl::SetMaxSelCount 修改范围。语句

```
m_wndCal.SetMaxSelCount (14);
```

把选中范围的上限设置成 14 天。作为增补的函数，GetMaxSelCount 返回当前最大选中数目。

如果要通过编程在多选月历控件中选中一个日期或一个日期段，您必须使用 CMonthCalCtrl::SetSelRange，而不能用 CMonthCalCtrl::SetCurSel。（如果在多选月历控件中调用后者，则会失败。）语句

```
m_wndCal.SetSelRange (CTime (1999, 9, 30, 0, 0, 0),
  CTime (1999, 9, 30, 0, 0, 0));
```

在 MCS_MULTISELECT 样式的日历控件中选中 1999 年 9 月 30 日，而语句

```
m_wndCal.SetSelRange (CTime (1999, 9, 16, 0, 0, 0),
  CTime (1999, 9, 30, 0, 0, 0));
```

选中 9 月 16 日到 9 月 30 日。如果您没有首先调用 SetMaxSelCount 把选中范围设置成 15 天或更大，则上述调用就会失败。如果要读取当前选中项，则用 CMonthCalCtrl::GetSelRange，如

下所示：

```
CTime dateStart, dateEnd;
m_wndCal.GetSelRange (dateStart, dateEnd);
```

这个示例将 dateStart 设置成选中的起始日期，dateEnd 设置成选中的结束日期。如果只选中了一天，dateStart 则等于 dateEnd。幸运的是，GetSelRange 不会遇到 GetCurSel 会碰见的那种随机"故障"。

三种月历控件样式允许您改变月历控件的外观。MCS_NOTODAY 清除日历底部显示今日日期的一行数据；MCS_NOTODAYCIRCLE 清除日历中围绕今日日期的那个圆；而 MCS_WEEKNUMBERS 显示星期数(1 到 52)。还可以用 CMonthCalCtrl 函数进一步修改日历的外观。例如：您可以用 SetToday 改变今天的日期（如控件显示的那样），用 SetFirstDayOfWeek 改变出现在日历左端一列的星期几，用 SetColor 改变控件的颜色。甚至可以命令控件调用它的 SetDayState 函数或处理 MCN_GETDAYSTATE 通知用黑体显示指定的日期。注意，只有控件样式中包含 MCS_DAYSTATE 时，SetDayState 才会起作用(才会有 MCN_GETDAYSTATE 通知发送)。

如果您想知道日历控件中当前日期(或日期范围)何时变化，可以处理两个通知的任一个。MCN_SELECT 通知是在用户选中新日期或日期段时发送的。MCN_SELCHANGE 通知则是在用户显式作出选择和因为用户将日历拨前或拨后一个月以改变选中日期时发送的。在 MFC 应用程序中，可以通过 ON_NOTIFY 将这些消息映射成父窗口类中的成员函数，或通过 ON_NOTIFY_REFLECT 将它们反射回派生控件类的函数中。

16.5.4 日期-时间拾取控件

日期-时间拾取控件，或称为"DTP 控件"，为开发者提供了一种简单、方便和易用的方法向用户请求日期和时间。DTP 控件和编辑控件相似，但是它显示的不是普通文本字符串，它显示的是日期和时间。日期可以用短格式显示，如 9/30/99，也可以用长格式显示，如"星期四，1999 年 9 月 30 日"。在标准格式(时：分：秒)中时间后面跟有 AM 或 PM。控件还支持自定义日期和时间格式。通过单击控件的向上和向下箭头，或在下拉月历控件中进行选择，用户可以看着编辑时间和日期。自然也可以动手输入时间和日期。MFC 通过封装类 CDateTimeCtrl 简化了 DTP 控件的接口。

用 DTP 控件请求输入时间只需要一两行代码。首先赋予控件样式 DTS_TIMEFORMAT，将它设置成只能显示时间。然后调用 CDateTimeCtrl::SetTime 设定显示在控件中的时间，并在准备提取时间时调用 CDateTimeCtrl::GetTime。假定 m_wndDTP 是对话框类中的一个 CDateTimeCtrl 数据成员，m_wndDTP 被映射到对话框中的 DTP 控件，下列 OnInitDialog 和 OnOK 函数在控件和对话框类的 CTime 成员变量间传输数据：

```
// In CMyDialog's class declaration
CTime m_time;
  .
  .
  .
BOOL CMyDialog::OnInitDialog()
{
  CDialog::OnInitDialog();
  m_wndDTP.SetTime(&m_time);
  return TRUE;
}

void CMyDialog::OnOK()
{
  m_wndDTP.GetTime(m_time);
  CDialog::OnOK();
}
```

您可以在对话框的 DoDataExchange 函数中添加一个 DDX_DateTimeCtrl 语句来代替显式调用 SetTime 和 GetTime：

```
DDX_DateTimeCtrl(pDX, IDC_DTP, m_time);
```

如果您使用 DDX_DateTimeCtrl 把 DTP 控件和对话框数据成员关联起来，您可能还需要使用 MFC 的 DDV_MinMaxDateTime 函数对从控件提取的时间进行合法性检查。

如果要在 DTP 控件中显示日期而不显示时间，则用短日期格式 DTS_SHORTDATEFORMAT 或长日期格式 DTS_LONGDATEFORMAT 取代 DTS_TIMEFORMAT。同设定和提取时间一样，您可以用 SetTime 和 GetTime 或 DDX_DateTimeCtrl 设定和提取日期。还可以使用 CDateTimeCtrl::SetRange 限定 DTP 控件可以接受的日期和时间。

样式包含 DTS_UPDOWN 的 DTP 控件具有向上和向下箭头，用户可用来编辑时间和日期。如果从控件样式中取消 DTS_UPDOWN，则用一个和组合框中箭头相似的向下箭头取代向上和向下箭头。单击该箭头会显示一个下拉月历控件，如图 16-14 所示。因此，将日期样式（DTS_SHORTDATEFORMAT 或 DTS_LONGDATEFORMAT）和 DTS_UPDOWN 组合起来就能创建一种 DTP 控件，在这种控件中可以用向上和向下箭头输入日期；对于未使用样式 DTS_UPDOWN 创建的控件，其日期要从月历中拾取。在默认方式下，DTP 控件中拉下的月历和控件是左对齐的。通过在控件样式中包含 DTS_RIGHTALIGN 可以改变对齐形式。也可以使用样式 DTS_APPCANPARSE 允许用户手动编辑显示在 DTP 控件中的日期。即使没有这个样式，也可以用键盘的箭头键编辑输入的时间和日期。

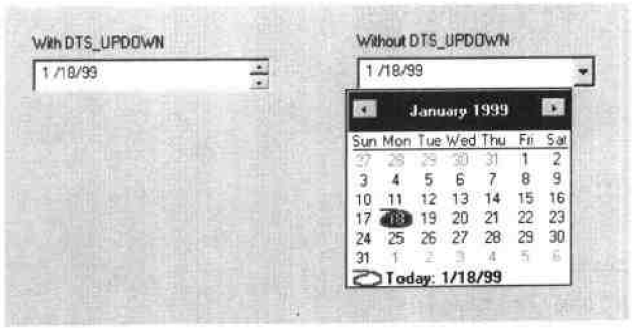

图 16-14 具有和不具有样式 DTS_UPDOWN 的日期-时间拾取控件

CDateTimeCtrl 的 SetFormat 函数把自定义的格式化字符串赋给 DTP 控件。例如：把形如"H':'mm':'ss"的格式化字符串赋给 DTP 控件，就可以用 24 小时标准格式显示时间。可以这样使用 SetFormat 和格式化字符串：

```
m_wndDTP.SetFormat (_T ("H\':\'mm\':\'ss"));
```

在格式化字符串中，H 代表 24 小时制格式的一位或两位小时数，mm 代表两位的分钟数，而 ss 代表两位的秒数。表 16-11 给出了格式化字符串中所有可用的特殊字符。通过把文字括在单引号中，还可以在格式化字符串中包含文字，比如上个例子中的冒号。如果想搞得更特别些，可以用 Xs 定义"回调域"。DTP 控件根据 DTN_FORMAT 和 DTN_FORMATQUERY 通知确定回调域中的显示内容，使处理这些通知的应用程序可以在运行过程中给 DTP 控件提供文字。

表 16-11 DTP 格式化字符

字符	说　明
d	一位或两位十进制日期
dd	两位数十进制日期
ddd	星期的三字符简写形式(例如 Mon 或 Tue)
dddd	星期的全写形式(例如 Monday 或 Tuesday)
h	12 小时制的一位或两位十进制小时数
hh	12 小时制的两位十进制小时数
H	24 小时制的一位或两位十进制小时数
HH	24 小时制的两位十进制小时数
m	一位或两位十进制分钟数
mm	两位十进制分钟数
M	一位或两位十进制月份数
MM	两位十进制月份数

续表

字符	说　明
MMM	月份的三字符简写形式(例如 Jan 或 Feb)
MMMM	月份的全写形式(例如 January 或 February)
s	一位或两位十进制秒数
ss	两位十进制秒数
t	显示 A 代替 a.m.,或 P 代替 p.m.
tt	显示 AM 代替 a.m.,或 PM 代替 p.m.
X	回调域
y	一位十进制年份数
yy	两位十进制年份数
yyyy	四位十进制年份数

DTP 控件给它的父窗口发送各种通知。如果要想知道何时显示下拉月历控件,请等候 DTN_DROPDOWN 通知。当 DTN_DROPDOWN 通知来到时,可以调用 CDateTimeCtrl::GetMonthCalCtrl 获得一个可用来修改月历控件的 CMonthCalCtrl 指针。如果只想知道 DTP 控件的时间和日期何时发生变化,则处理 DTN_DATETIMECHANGE 通知就能达到目的。如果想详细了解 DTP 控件通知,请查阅 Platform SDK 说明文档。

第 17 章　线程和线程同步化

在 Microsoft Win32 环境中，每个运行的应用程序都构成一个“进程”，而每个进程都包含一个或多个执行线程。“线程”指的是程序代码的执行途径，外加一组操作系统分配的资源（堆栈，寄存器状态等等）。

16 位和 32 位 Microsoft Windows 版本的根本区别就在于 32 位 Windows 没有每个应用程序只许有一个线程的限制。32 位 Windows 应用程序的进程是从单个线程开始的，但是线程还可以繁衍另外的线程。操作系统内核中区分优先级别的调度器在活动的线程间分配 CPU 时间，使它们看上去像在同时运行。对于既要在前端处理用户的输入，同时又要执行后台任务的工作，线程是一种理想的选择。它们也可起到可见的作用，像主线程处理发送给应用程序主窗口的消息那样，线程也可以创建窗口并处理发送给这些窗口的消息。

并不是每个人都能编写多线程应用程序。多线程应用程序难于编写和调试，因为同时运行多个线程所要求的并行性给程序的编写添加了一层困难。但是如果使用得当，多线程可以极大地提高应用程序的响应能力。例如：在专门的线程中执行拼写检查的字处理程序可以在主线程中继续处理消息，并且在拼写检查程序的运行过程中允许用户连续不断地工作。编写以线程方式执行的拼写检查程序，困难在于拼写检查线程要不断地与应用程序中的其他线程取得同步。大多数程序员都习惯于在同步条件下考虑他们的程序：函数 A 调用函数 B，函数 B 执行完某些任务后返回，等等。但是本质上线程是异步的。在多线程应用程序中，就必须考虑如下情况的发生：有两个线程同时调用函数 B，或是一个线程从它那里读取变量值而另一个线程对它写入变量值。如果函数 A 在另一个线程中启动了函数 B，那么还必须预见到如果函数 B 执行时而函数 A 仍继续执行而导致的问题。例如：将函数 A 在堆栈中创建的变量的地址传递给函数 B 进行处理是普遍的操作。但是如果函数 B 在另外一个线程中，那么等函数 B 该访问它时变量可能已经不存在了。即使看上去非常正确的程序，当涉及到使用不同的线程时都可能会存在致命的缺陷。

MFC 在 CWinThread 类中封装了可执行线程。它还在易于使用的 C++ 类中封装了事件、互斥和其他 Win32 线程同步对象。MFC 使多线程编程更简单了吗？并不完全这样。那些用 C 语言编写过多线程 Windows 应用程序的开发者经常奇怪地发现用 MFC 反而增加了复杂性。在 MFC 中编写多线程程序的关键是确实知道自己想要做什么、麻烦会出在哪儿。本章内容在这两个方面都会对您有所帮助。

17.1　线程

对于 Windows 来说，所有线程都是一样的。但是 MFC 却把线程区分为两个种类型：User Interface(UI)threads(用户界面(UI)线程)和 Worker threads(工作者线程)。两类线程的不同之处在于 UI 线程具有消息循环而工作者线程没有。UI 线程可以创建窗口并处理发送给这些窗口的消息。工作者线程执行后台任务，因其不接收用户的直接输入，所以不需要窗口和消息循环。

系统本身就提供了两个非常好的例子，可以说明 UI 线程和工作者线程的用途。当您在操作系统的命令解释器中打开文件夹时，命令解释器会启动一个 UI 线程创建一个窗口来显示文件夹中的内容。如果您将一组文件拖动复制到一个新的文件夹，该文件夹的线程就会执行文件传送任务。(有时 UI 线程会生成另一个线程，此时就是工作者线程，来复制文件。)这种多线程体系结构的优点在于，一旦开始复制，您就可以切换到其他文件夹打开的窗口中了，文件在后台中被复制的同时可以继续进行其他工作。启动一个 UI 线程创建窗口在概念上与在应用程序中启动一个应用程序相似。UI 线程的最常见的用法是创建多个由各自的执行线程服务的窗口。

工作者线程非常适合于执行那些可以从应用程序的其他部分中分离的独立任务以及在后台中执行的任务。工作者线程的一个典型的例子是用来播放 AVI 剪辑的动画控件。该线程的工作仅仅是绘制每帧图像，让自己睡眠一段极短的时间，然后苏醒再重新执行任务。它几乎没有给处理器添加工作量，因为每帧之间的暂停用去了它的大部分时间，可它却提供了可贵的服务。这是一个多线程设计的非常好的例子，后台线程被指派来完成特定的任务，然后开始反复执行它直到上线程通知结束为止。

17.1.1　创建工作者线程

在 MFC 应用程序中启动一个线程的最好方法是调用 AfxBeginThread。MFC 定义了两个不同的 AfxBeginThread 版本：一个启动 UI 线程，另一个启动工作者线程。在 Thrdcore.cpp 中可以找到它们的源程序代码。在 MFC 程序中，只有在线程不使用 MFC 时才可使用 Win32::CreateThread 函数来创建线程。AfxBeginThread 并不仅仅是 Win32::CreateThread 函数的封装；除了启动线程以外，它还要初始化主结构使用的内部状态信息，在线程创建过程中的不同地方执行合理的检查，并采用一定的方法来确保以线程安全的方式访问 C 运行时库中的函数。

AfxBeginThread 使创建工作者线程变得非常简单，几乎微不足道。当调用 AfxBeginThread 时，它将创建一个新的 CWinThread 对象，启动一个线程并使其附属于 CWinThread 对象，返回一个 CWinThread 指针。语句

```
CWinThread * pThread = AfxBeginThread(ThreadFunc, &threadInfo);
```

启动一个工作者线程并给它传递一个应用程序定义的数据结构的地址(&threadInfo),其中包含了对线程的输入。ThreadFunc 是"线程函数",这类函数在线程开始执行后才得以执行。下面给出一个非常简单的线程函数,它插入循环占用部分 CPU 时间,然后结束:

```
UINT ThreadFunc (LPVOID pParam)
{
 UINT nIterations = (UINT) pParam;
 for (UINT i = 0; i < nIterations; i++);
 return 0;
}
```

在本例中,pParam 内传递的值不再是一个指针而是普通的 UINT。在下一节中将详细讨论线程函数。

AfxBeginThread 的工作者线程可以接收另外的 4 个参数,分别用来指定线程的优先级别、堆栈尺寸、产生标志以及安全属性。函数的完整原型是:

```
CWinThread* AfxBeginThread (AFX_THREADPROC pfnThreadProc,
 LPVOID pParam, int nPriority = THREAD_PRIORITY_NORMAL,
 UINT nStackSize = 0, DWORD dwCreateFlags = 0,
 LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL)
```

nPriority 指定了线程的执行优先级别。高优先级的线程在低优先级的线程之前得到 CPU 时间,实际上即使是最低优先级别的线程通常也可以得到所需要的处理器时间。nPriority不是指定绝对的优先级别。其优先级别是相对于该线程所属的进程优先级别而指定的。默认值为 THREAD_PRIORITY_NORMAL,此时会为此线程分配与拥有它的进程相同的优先级别。使用 CWinThread::SetThreadPriority 可以在任何时候修改线程的优先级别。

传递给 AfxBeginThread 的 nStackSize 参数指定了线程最大的堆栈尺寸。在 Win32 环境下,每个线程都可以接收自己的堆栈。默认 nStackSize 值 0 允许堆栈增加到 1MB 大小。这并不是意味着每个线程都要求至少 1MB 的内存,只是说在 32 位 Windows 应用程序执行时所用到的 4GB 地址空间中给每个线程都分配了 1MB 的地址空间。如果不需要的话,内存不会分配地址空间给堆栈,所以大多数线程堆栈从来不会使用超出几千字节的实际内存。用一个值来限制堆栈的尺寸可使操作系统捕获失控的函数,这些函数会无穷尽地反复执行,最终耗尽堆栈。默认限制 1 MB,几乎对于所有的应用程序都比较合适。

dwCreateFlags 可以是两个值中的任一个。默认值 0 告诉系统立即开始执行线程。如果指定了 CREATE_SUSPENDED,线程开始时就处于暂停状态,直到另一个线程(通常是创建它的线程)在暂停的线程上调用了 CWinThread::ResumeThread 之后才会继续运行,例如:

```
CWinThread* pThread = AfxBeginThread (ThreadFunc, &threadInfo,
 THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
```

```
.
.
.
pThread->ResumeThread(); // Start the thread
```

有时创建一个线程而将其延迟到以后再执行是很有用的。CREATE_SUSPENDED 标志就是规定延迟执行的机制。

在 AfxBeginThread 参数列表中的最后一个参数 lpSecurityAttrs 是指向 SECURITY_ATTRIBUTES结构的指针，该结构指定了新线程的安全属性，并告诉系统子进程是否继承了线程句柄。默认值 NULL 意味着新线程与创建它的线程具有相同的属性。

线程函数

线程函数是回调函数，因此它必须是静态成员函数或是在类外部声明的全局函数。其原型如下：

```
UINT ThreadFunc (LPVOID pParam)
```

pParam 是一个 32 位值，等于传递给 AfxBeginThread 的 pParam。通常，pParam 是应用程序定义的数据结构的地址，该结构包含了由工作者线程的创建线程传递给工作者线程的信息。它还可以是一个标量或句柄，甚至是指向对象的指针。对于两个以上的线程使用线程函数是合法的，但是应该小心由全局和静态函数造成的重入问题。只要线程使用的变量(和对象)是在堆栈上创建的，就不会发生重入问题，因为每个线程都拥有自己的堆栈。

17.1.2　创建 UI 线程

创建 UI 线程与创建工作者线程具有截然不同的过程。工作者线程是由其线程函数定义的，而 UI 线程的行为却是由从 CWinThread 派生来的可动态创建类控制的，该类与从 CWinApp 派生的应用程序类很相似。下面给出的 UI 线程类创建了一个顶层框架窗口，用鼠标左键单击时窗口会自动关闭。关闭窗口的同时也结束了线程，因为 CWnd::OnNcDestroy 会给线程的消息队列发送一个 WM_QUIT 消息。给主线程发送一个 WM_QUIT 消息可以结束线程以及应用程序。

```
// The CUIThread class
class CUIThread : public CWinThread
{
    DECLARE_DYNCREATE(CUIThread)

public:
    virtual BOOL InitInstance();
};

IMPLEMENT_DYNCREATE(CUIThread, CWinThread)
```

```
BOOL CUIThread::InitInstance()
{
    m_pMainWnd = new CMainWindow;
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}

// The CMainWindow class
class CMainWindow : public CFrameWnd
{
public:
    CMainWindow();

protected:
    afx_msg void OnLButtonDown(UINT, CPoint);
    DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(CMainWindow, CFrameWnd)
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()

CMainWindow::CMainWindow()
{
    Create(NULL, _T("UI Thread Window"));
}

void CMainWindow::OnLButtonDown(UINT nFlags, CPoint point)
{
    PostMessage(WM_CLOSE, 0, 0);
}
```

注意,这里用 sw_show 参数代替常用的 m_nCmdShow 参数传递给了 ShowWindow。m_nCmdShow是 CWinApp 的数据成员,所以在使用 UI 线程创建顶层窗口时,您需要指定窗口的初始状态。

通过调用 AfxBeginThread 来启动 CUIThread,前者接收指向线程类的 CRuntimeClass 指针:

```
CWinThread* pThread = AfxBeginThread(RUNTIME_CLASS(CUIThread));
```

AfxBeginThread 的 UI 线程版本接收与工作者线程版本中相同的 4 个可选参数,但不接收 pParam 值。一旦启动,UI 线程就会与创建它的线程异步运行。

17.1.3 暂停和继续执行线程

运行中的线程可以用 CWinThread::SuspendThread 暂停,再用 CWinThread::ResumeThread

继续执行。线程可以为自己调用 SuspendThread，也可以是别的线程为它调用 SuspendThread。但是暂停的线程不能够自己调用 ResumeThread 恢复执行；必须是其他线程为它调用 ResumeThread。暂停的线程不会消耗处理器的时间，并且给系统增加的开销几乎等于零。

对于每个线程，系统都维持着一个“暂停数”，其值由 SuspendThread 加 1、由 ResumeThread 减 1。只有在其暂停数为 0 时，才会给线程调度处理器时间。如果 SuspendThread 被连续地调用了两次，ResumeThread 也必须被调用两次。没有用 CREATE_SUSPENDED 标志创建的线程初始暂停数为 0。用 CREATE_SUSPENDED 标志创建的线程开始就具有暂停数 1。SuspendThread和 ResumeThread 都返回线程以前的暂停数，这样您就可以确保线程被继续执行了。无论暂停数有多大都可以反复调用 ResumeThread 直到其返回值为 1。如果线程当前没有被暂停，ResumeThread 就会返回 0。

17.1.4　使线程睡眠

通过调用 API 函数::Sleep，线程可以让自己睡眠指定的一段时间。正在睡眠的线程不占用处理器时间。语句

```
::Sleep(10000);
```

使当前线程暂停 10 秒钟。

::Sleep 的一种用处是实现那些本质上是基于时间的线程，例如：动画控件中的背景线程或移动时钟指针的线程。::Sleep 还可以用来放弃剩余的线程时间片。语句

```
::Sleep(0);
```

暂停当前线程并允许调度程序运行其他具有相同或更高的优先级别的线程。如果没有其他优先级相同或更高的线程处于等待状态，那么函数调用就立即返回并继续执行当前的线程。在 Microsoft Windows NT 4.0 及更高版本中，通过调用::SwitchToThread 可以执行另一个线程。如果编写的程序必须运行在所有 Win32 平台上，就要使用 ::Sleep (0)。

如果编写一个使用多线程在屏幕上绘图的应用程序，有策略地使用几个::Sleep (0) 语句可以惊人地提高输出质量。假设您要设计 4 个对象的动画动作，给每个对象都分配了一个线程。如果只是简单地在一个循环中运行每个线程，让它得到它所能争取到的所有处理器时间，那么对象的运动就可能会很粗糙没有规则。但是，如果让每个线程都将分配给它的对象每次移动几个像素的距离，然后就调用::Sleep (0)，那么动画的执行效果就会更平滑。

传递给::Sleep 的值并不能保证线程在指定的间隔时间过去后的某个精确时刻醒过来。给::Sleep 传递一个 10 000 的值只能保证线程将在 10 秒过后的某个时刻醒转。线程可能睡眠 10 秒，也可能睡眠 20 秒，这要取决于操作系统。在实际中，通常线程会在指定的间隔时间过去之后很短的时间内继续运行。现在，在所有 Windows 版本中还不存在能够以精确的时间来暂停线程的方法。

17.1.5 终止线程

线程开始执行后,有两种方法可以终止它。当线程函数执行 return 语句时,或是此线程中任何地方的任何函数调用 AfxEndThread 时,工作者线程就会结束。当给其消息队列发送了 WM_QUIT 消息或是线程自己调用了 AfxEndThread 时,UI 线程就会结束。使用 API 函数 ::PostQuitMessage,UI 线程可以给自己发送 WM_QUIT 消息。AfxEndThread、::PostQuitMessage 和 return 都接受一个 32 位的出口代码,在线程结束之后可以用::GetExitCodeThread 检索得到。下列语句将由 pThread 引用的线程出口代码复制给了 dwExitCode:

```
DWORD dwExitCode;
::GetExitCodeThread (pThread->m_hThread, &dwExitCode);
```

如果对正在执行的线程调用了该函数,::GetExitCodeThread 就会将 dwExitCode 设置为 STILL_ACTIVE (0x103)。在本例中,传递给::GetExitCodeThread 的句柄是从封装线程的 CWinThread 对象的数据成员 m_hThread 中得到的。如果有 CWinThread 对象并希望调用要求一个线程句柄的 API 函数,那么都可以从 m_hThread 中得到这个句柄。

17.1.6 自动删除 CWinThread

上节中的两行程序代码看上去非常正确,而实际上它们却是即将发生的事故,除非意识到了 CWinThread 所具有的某些特殊的性质并采取了具体行动去处理它。

已经知道 AfxBeginThread 创建一个 CWinThread 对象并将其地址返回给调用者。但是怎样删除 CWinThread 呢? 没必要必须通过由 AfxBeginThread 返回的 CWinThread 指针来调用 delete,因为 MFC 会在线程结束后通过指针来调用 delete。而且。CWinThread 的析构函数会使用::CloseHandle API 函数来关闭线程句柄。线程句柄必须以显式的方式被关闭,因为即使与句柄相关的线程终止了,它们也仍然会处于打开状态。它们必须保持为打开状态,否则像::GetExitCodeThread 这样的函数就不可能工作了。

从表面上看,MFC 自动删除 CWinThread 对象并关闭相应的线程句柄好像很方便。如果 MFC 不处理这些内务,您就必须亲自处理它们。但是这里存在一个问题,至少是潜在一个问题。让我们再看一下语句:

```
::GetExitCodeThread (pThread->m_hThread, &dwExitCode);
```

如果线程没有结束,这行代码丝毫没有错误,因为 pThread 仍然是有效的指针。但是如果线程已经结束,那么 MFC 就很可能已经删除了 CWinThread 对象,现在 pThread 就是无效指针了。(我说"很可能"是因为在线程结束和关联的 CWinThread 对象删除之间有一段短暂的时间。)一个明了的解决办法是在线程结束之前将线程句柄从 CWinThread 对象复制到本地变量中,并在对::GetExitCodeThread 的调用中使用此句柄,如下所示:

```
// While the thread is running
HANDLE hThread = pThread->m_hThread;
  .
  .
  .
// Sometime later
::GetExitCodeThread(hThread, &dwExitCode);
```

但是此程序也是错误的，为什么？因为如果 CWinThread 对象不再存在的话，线程句柄也不会存在；它早已经被关闭了。如果您使用像::GetExitCodeThread 这样的函数，即使线程不再运行了它还假定线程的句柄仍然有效，而没有考虑到 CWinThread 的自动删除特性和 CWinThread 的析构函数所执行的::CloseHandle 调用，那么就会导致严重的程序设计错误。

幸运的是，实际上此问题有两种解决办法。通过设置对象的 m_bAutoDelete 数据成员为 FALSE 可防止 MFC 删除 CWinThread 对象。默认值是 TRUE 允许自动删除。如果您选择这种方法，记住要通过 AfxBeginThread 返回的 CWinThread 指针来调用 delete，否则您的应用程序就可能因内存不足而无法运行。下列程序说明了这一点：

```
CWinThread* pThread = AfxBeginThread(ThreadFunc, NULL,
  THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
pThread->m_bAutoDelete = FALSE;
pThread->ResumeThread();
  .
  .
  .
// Sometime later
DWORD dwExitCode;
::GetExitCodeThread(pThread->m_hThread, &dwExitCode);
if (dwExitCode == STILL_ACTIVE) {
  // The thread is still running.
}
else {
  // The thread has terminated. Delete the CWinThread object.
  delete pThread;
}
```

创建线程先使其处于暂停状态与删除 CWinThread 对象同样重要。如果不这样做，一种发生的可能性很小但是确实存在的事情就会出现：在创建它的线程设置 m_bAutoDelete 为 FALSE 之前，新的线程已经结束它的生命期了。要记住：一旦启动了线程，Windows 就无法保证给线程分配多少 CPU 时间了。

第二个解决办法是允许 CWinThread 执行自动删除，但是要使用 Win32 ::DuplicateHandle 函数创建一个线程句柄的复件。线程句柄要进行引用计数，使用::DuplicateHandle 复制一个

新打开的线程句柄会将引用数从 1 增加到 2。所以,当 CWinThread 的析构函数调用::CloseHandle 时,句柄实际上并没有被关闭;仅仅是递减了它的引用计数。这种方法的缺陷是必须亲自调用::CloseHandle 来关闭句柄。示例如下:

```
CWinThread* pThread = AfxBeginThread (ThreadFunc, NULL,
  THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);

HANDLE hThread;
::DuplicateHandle (GetCurrentProcess (), pThread->m_hThread,
  GetCurrentProcess (), &hThread, 0, FALSE, DUPLICATE_SAME_ACCESS);

pThread->ResumeThread ();
  .
  .
  .
// Sometime later
DWORD dwExitCode;
::GetExitCodeThread (hThread, &dwExitCode);
if (dwExitCode == STILL_ACTIVE) {
  // The thread is still running.
}
else {
  // The thread has terminated. Close the thread handle.
  ::CloseHandle (hThread);
}
```

注意,要在暂停状态下创建新线程,以便绝对地保证正在创建的线程可以在它结束之前执行。

17.1.7 结束另一个线程

一般而言,线程只能结束它们自己。如果您想让线程 A 结束线程 B,就必须建立一个发信机制,允许线程 A 告诉线程 B 结束执行。一个简单的变量可以作为终止请求标志,说明如下:

```
// Thread A
nContinue = 1;
CWinThread* pThread = AfxBeginThread (ThreadFunc, &nContinue);
  .
  .
  .
nContinue = 0; // Tell thread B to terminate.

// Thread B
```

```
UINT ThreadFunc (LPVOID pParam)
{
  int * pContinue = (int *) pParam;
  while (*pContinue) {
    // Work work work work
  }
  return 0;
}
```

在本例中线程 B 要经常检查 nContinue，如果 nContinue 从非零值变为 0 则结束线程。通常让两个线程在没有同步化的情况下访问同一个变量并不是很好的主意，但是在本例中可以这样做，因为线程 B 只是进行检查来确定 nContinue 是否为 0。当然为了避免非法访问，您需要确保在线程 B 运行时 nContinue 没有超出有效范围。把 nContinue 设置为静态或全局变量就可以。

现在假设您想要修改本例，使线程 A 将 nContinue 将设置为 0，然后它就暂停直到线程 B 不再运行为止。下面给出一种实现此目的的合适方法：

```
// Thread A
nContinue = 1;
CWinThread * pThread = AfxBeginThread (ThreadFunc, &nContinue);
  .
  .
  .
HANDLE hThread = pThread->m_hThread; // Save the thread handle.
nContinue = 0; // Tell thread B to terminate.
::WaitForSingleObject (hThread, INFINITE);

// Thread B
UINT ThreadFunc (LPVOID pParam)
{
  int * pContinue = (int *) pParam;
  while (*pContinue) {
    // Work work work work
  }
  return 0;
}
```

::WaitForSingleObject 阻止调用线程直到指定的对象（在本例中是另一个线程）进入“信号发出”状态。一个线程在结束后就会处于信号发出状态。在::WaitForSingleObject 中线程暂停时，它会处于有效的等待状态，因为它是等待着直到调用函数返回。在本例中假定了线程 B 在线程 A 通知它之前不会结束。如果情况不是这样，即如果线程 B 在线程 A 命令它结束以前可以终止，线程 A 就会在暂停状态下创建线程 B 并用::DuplicateHandle 复制一个线程句

柄。否则,线程 A 就会陷入给::WaitForSingleObject 传递了一个无效线程句柄的麻烦。

::WaitForSingleObject 是一个必不可少的函数,您在编写多线程程序时会多次用到它。传递给它的第一个参数是欲使其等待的对象的句柄。(还可以是一个进程句柄、同步化对象句柄或文件修改通知句柄,等等。)在上例中,线程 A 在把 nContinue 设置为 0 之前得到了线程 B 的句柄,因为代表线程 B 的 CWinThread 对象在执行对::WaitForSingleObject 的调用时可能已经不存在了。在::WaitForSingleObject 中的第二个参数是希望等待的时间长度。当指定了 INFINITE 后,如果正在等待的对象永远不会进入信号发出状态,那么就有机会封锁调用线程了。但是如果指定了一个毫秒值,如下:

```
::WaitForSingleObject (hThread, 5000);
```

::WaitForSingleObject 就会在指定的时间用完之后(这里是 5 秒)返回,即使对象还没有进入信号发出状态。您可以通过检查返回值来确定函数返回的原因。WAIT_OBJECT_0 意味着对象进入了信号发出状态,而 WAIT_TIMEOUT 表示还没有进入。

给定一个线程句柄或有效地封装了线程句柄的 CWinThread 对象,通过调用::WaitForSingleObject并将等待时间指定为 0 可以快速确定线程是否仍在运行,示例如下:

```
if (::WaitForSingleObject (hThread, 0) == WAIT_OBJECT_0) {
  // The thread no longer exists.
}
else {
  // The thread is still running.
}
```

以这种方式调用,::WaitForSingleObject 就不会等待而直接返回。返回值为 WAIT_OBJECT_0 意味着线程进入了信号发出状态(不再存在),返回值等于 WAIT_TIMEOUT 说明线程没有进入信号发出状态(仍然存在)。通常是由用户来确保传递给::WaitForSingleObject 的句柄有效,这可以通过复制原始线程句柄或者防止 CWinThread 被自动删除等方式来完成。

有一种方法线程可以用来直接终止另一个线程,但是只能把它作为最后一着。语句

```
::TerminateThread (hThread, 0);
```

结束句柄为 hThread 的线程并将一个退出代码 0 赋给它。Win32 API 参考文献中列出了一些::TerminateThread 可能会造成的问题,从孤立线程同步对象到无法正常结束的 DLL。

17.1.8 线程、进程以及优先级别

调度程序是操作系统的组成部分,它决定了哪个线程运行以及运行多长时间。线程调度是很复杂的任务,它的主要目标就是在多个执行的线程之间尽可能高效地分配 CPU 时间,产生一种好像同时运行多个线程的错觉。在具有多个 CPU 的机器上,Windows NT 和

Windows 2000 实际上同时运行了两个以上的线程，它们采用了一种称为“对称多处理”（或称为 SMP）的策略将不同的线程分配给了不同的处理器。Windows 95 和 Windows 98 不是 SMP 操作系统，因此它们在同一个 CPU 上调度所有线程，即使在具有多个处理器的 PC 机上。

调度程序使用了多种技术来改进多任务的执行效率，并努力确保系统中的每个线程都能获得足够的 CPU 时间。（想了解 Windows NT 内在的调度程序、它的策略以及算法，我可以推荐一本好书：David Solomon 所著的《Inside Windows NT》第二版。）但是，最终决定要执行的下一个线程是具有最高优先级的线程。在任何时刻，每个线程都分配了一个从 0 到 31 的优先级。如果优先级为 11 的线程正在等待执行，而所有其他竞争 CPU 时间的线程具有的优先级都为 10 或更小，那么下一个执行的就是优先级为 11 的线程。如果有两个优先级都为 11 的线程在等待执行，调度程序将执行最近最少执行的一个。当线程的时间片（或称为“时间量子”）用完之后，如果所有其他线程仍然具有较低的优先级，那么就会执行另一个优先级为 11 的线程。按照规律，调度程序总是将时间片分配给等待线程中具有最高优先级别的线程。

那么这意味着低优先级别的线程永远不会得到执行吗？并非如此。首先，要知道 Windows是一个基于消息的操作系统。如果线程调用::GetMessage 而它的消息队列是空的，那么该线程就会停止直到再次有了可用的消息。这就给了低优先级线程执行的机会。大多数 UI 线程把它们的大部分时间用在等待消息的暂停状态上了，因此只要高优先级别的工作者线程不垄断 CPU，即使是具有最低优先级别的线程也会得到它们所需的全部时间。（工作者线程从来不会暂停在消息队列上，因为它们根本不处理消息。）

调度程序还在优先级别上施加了许多技巧，提高了系统整体的响应能力并减少了线程得不到 CPU 时间的机会。如果一个优先级为 7 的线程已经很长一段时间没有接收到时间片了，调度程序可能就会临时将它的优先级别提高到 8 或 9 甚至更高，给它一个执行的机会。Windows NT 3.x 会提高属于前台进程的线程的优先级别，从而改善了用户正在使用的应用程序的响应能力，而 Windows NT 4.0 Workstation 会增加线程的时间量子。Windows 还使用了一种称为“优先级继承”的技术，避免具有高优先级的线程在低优先级线程所拥有的同步对象上暂停的时间太长。例如：如果优先级别为 11 的线程要求一个由优先级别为 5 的线程拥有的互斥对象，调度程序可能就会提高优先级别为 5 的线程的优先级别，以便很快地提供互斥对象。

最初线程优先级是如何分配的呢？当您调用 AfxBeginThread 或 CWinThread::SetThreadPriority 时，要指定“相对线程优先级”。操作系统会结合相对优先级和拥有线程的进程的优先级（类型稍后详述）计算出线程的“基本优先级”。实际运行中线程的优先级（编号从 0 到 30）由于被提高或取消提高，所以在不同的时刻也不同。虽然不能提高线程的优先级（即使可以，您也不会那么做的），但是您可以通过设置进程的优先级类型和相对线程优先级来控制基本优先级别。

进程优先级别类型

大多数进程开始时都具有优先级类型 NORMAL_PRIORITY_CLASS，但是一旦启动，进程就可以调用::SetPriorityClass 来修改它的优先级，该函数接受进程句柄(可用::GetCurrentProcess 获得)和表 17-1 给出的参数。

表 17-1 进程优先级别类型

优先级别类型	说 明
IDLE_PRIORITY_CLASS	只有在系统处于空闲时进程才运行，例如：对于给定的 CPU 没有其他线程正在等待时
NORMAL_PRIORITY_CLASS	默认的进程优先级别类型。进程不需要特殊的调度
HIGH_PRIORITY_CLASS	进程接收的优先级别在 IDLE_PRIORITY_CLASS 和 NORMAL_PRIORITY_CLASS 进程之上
REALTIME_PRIORITY_CLASS	进程必须具有可能的最高优先级，它的线程应该比甚至是属于 HIGH_PRIORITY_CLASS 进程的线程具有更高的优先级

大多数应用程序不需要修改它们的优先级类型。HIGH_PRIORITY_CLASS 和 REALTIME_PRIORITY_CLASS 进程会极大地抑制系统的响应能力，甚至会延迟关键的系统行为，如清除磁盘高速缓冲区。HIGH_PRIORITY_CLASS 的一个合法的用途是用于系统应用程序，大部分时间它都隐藏起来，只有当某种输入事件发生时它才弹出一个窗口。这些应用程序在它们暂停等待输入时只占系统极少的额外开销，但是一旦有某种输入出现，它们就会获得比一般应用程序高的优先级。REALTIME_PRIORITY_CLASS 主要是为实时数据获取程序提供的，为了能够适当地工作它们必须共享 CPU 时间。IDLE_PRIORITY_CLASS 很适合于屏幕保护、系统监视以及其他低级应用程序，它们主要用来在后台执行不引人注目的操作。

相对线程优先级

表 17-2 给出了相对线程优先级的值，可以把它们传递给 AfxBeginThread 和 CWinThread::SetThreadPriority。默认值为 THREAD_PRIORITY_NORMAL，除非指定了其他值否则 AfxBeginThread 会自动把它赋给线程。通常，属于 NORMAL_PRIORITY_CLASS 进程的 THREAD_PRIORITY_NORMAL 线程的基本优先级为 8。在不同的时刻，出于上面讨论过的某种原因线程的优先级可能会提高，但是它最终会回到 8。运行在 HIGH_PRIORITY_CLASS 后台或前端进程中的 THREAD_PRIORITY_LOWEST 线程具有的基本优先级为 11。实际的编号其实并不很重要，因为可以在一个进程中微调线程的优先级来实现最好的响应能力和运行效率，而且如果必要，还可以调整进程自身的优先级。

17-2　相对线程优先级

优先级的值	说　明
THREAD_PRIORITY_IDLE	如果进程的优先级类型为 HIGH_PRIORITY_CLASS 或更低，则线程的基本优先级就为 1。如果进程的优先级类型为 REALTIME_PRIORITY_CLASS，则基本优先级就为 16
THREAD_PRIORITY_LOWEST	线程的基本优先级等于进程的优先级类型减 2
THREAD_PRIORITY_BELOW_NORMAL	线程的基本优先级等于进程的优先级类型减 1
THREAD_PRIORITY_NORMAL	默认的线程优先级值。线程的基本优先级等于进程的优先级类型
THREAD_PRIORITY_ABOVE_NORMAL	线程的基本优先级等于进程的优先级类型加 1
THREAD_PRIORITY_HIGHEST	线程的基本优先级等于进程的优先级类型加 2
THREAD_PRIORITY_TIME_CRITICAL	如果进程的优先级类型为 HIGH_PRIORITY_CLASS 或更低，则线程的基本优先级就为 15，如果进程的优先级类型为 REALTIME_PRIORITY_CLASS，则基本优先级别就为 31

现在您已经理解了线程优先级的来源以及它们对调度进程的影响，接下来我们将讨论如何知道何时去调整线程的优先级以及应该给它们赋予什么样的值。一般规律是，如果要求高的优先级，那么理由通常是明确的。如果要求高优先级的理由不明确，那么使用普通的线程优先级就可以。对于大多数线程，默认值 THREAD_PRIORITY_NORMAL 就足够了。但是如果您正在编写一个应用程序，它使用专门的线程读取和缓冲从串行端口进入的数据，则除非读取和缓冲线程的相对优先级别值为 THREAD_PRIORITY_HIGHEST 或 THREAD_PRIORITY_TIME_CRITICAL，否则它将时不时地丢失字节。

可以保证一点：如果应用程序过多地占据了 CPU，但又不是为了实现某种特殊的目的，如专门用于在 PC 上执行实时数据获取任务，那么该程序在市场上就不会获得欢迎。CPU 时间是计算机的最宝贵的资源。要非常合理地使用它，不要犯这样的错误：为了使自己的应用程序执行速度增加百分之五而提高了优先级，结果却使得其他应用程序的速度和响应能力减少了百分之五十。

17.1.9　在多线程应用程序中使用 C 运行时函数

在多线程应用程序中使用标准 C 运行时库中的某些函数会造成问题。strtok、asctime 和几个其他 C 运行时函数用全局变量来保存中间数据。如果线程 A 要调用其中的一个函数，而线程 B 抢先于线程 A 调用了同一个函数，由线程 B 保存的全局数据就可能覆盖由线程 A 保存的全局数据，或者相反。这个问题的一个解决方法是使用线程同步化对象串行化对运行函数的访问。但是即使是简单的同步对象就占用处理器而言，代价也是昂贵的。因此，现

代的大多数 C 和 C++ 编译器都具有两种 C 运行时库版本：一种是线程安全型的(可以被两个以上的线程调用)而另一种不是。运行时库的线程安全型版本通常不依赖于线程的同步化对象。相反,它在每个线程的数据结构中保存中间结果。

Visual C++ 带有 6 种不同的 C 运行时库版本。选用它们的依据是：正在编译的程序是进行调试状态下的创建还是发布版本的创建;希望是静态链接 C 运行时库还是动态链接;以及应用程序是单线程还是多线程的。表 17-3 给出了库的名称和相应的编译器选择开关。

表 17-3 VISUAL C++ 中 C 运行时库版本

库名称	应用程序类型	选择开关
Libc.lib	单线程;静态链接;发布版本的创建	/ML
Libcd.lib	单线程;静态链接;调试状态下创建	/MLd
Libcmt.lib	多线程;静态链接;发布版本的创建	/MT
Libcmtd.lib	多线程;静态链接;调试状态下创建	/MTd
Msvcrt.lib	单线程或多线程;动态链接;发布版本的创建	/MD
Msvcrtd.lib	单线程或多线程;动态链接;调试状态下创建	/MDd

Libc.lib、Libcd.lib、Libcmt.lib 和 Libcmtd.lib 是包含 C 运行时程序的静态链接库;Msvcrt.lib 和 Msvcrtd.lib 是引入库,使应用程序可以动态地与 Visual C++ C 运行时 DLL 中的函数进行链接。当然,除非要创建自己的生成文件,否则大可不必过多地去理会编译器的选择开关。如果使用的是 Visual C++,只要在 Project Settings 对话框中的 Use Run-time Library 域中选择了合适的输入项,IDE 就会为您添加选项开关。即使您编写的是不用 C 运行时函数的多线程应用程序,也应该链接一个多线程库,因为 MFC 自己要调用某些 C 运行时函数。

在 MFC 应用程序中,要对 C 运行时函数进行线程安全型的调用,这就是要做的一切工作。只要设置编译器选择开关,就完全可以让类库来完成剩下的全部任务了。在 SDK 应用程序中,您还得用对_beginthreadex 的调用替换对::CreateThread 的调用。MFC 程序员不必去考虑_beginthreadex,因为 AfxBeginThread 会自动调用它。

17.1.10 跨线程界限调用 MFC 成员函数

现在来谈谈有关编写多线程 MFC 应用程序的坏消息。只要线程不调用其他线程创建的对象的成员函数,就几乎不存在对它们执行操作的限制。但是,如果线程 A 给线程 B 传递了一个 CWnd 指针而线程 B 要调用 CWnd 对象的成员函数,那么 MFC 在调试状态下可能就会出现断言错误。发布版本的创建可能会正常,但是也可能不会。还有一种可能性是调试状态下创建不会出问题,但是却不能正常地运行。这完全取决于在此特定的 CWnd 成员函数被调用时主结构内部所发生的事情。可以避免这种潜在的问题,通过划分线程来实现,使每个线程仅仅使用自己创建的对象而不是依赖于其他线程创建的对象。但是对于某些情

况此方法并不实际，下面将给出一些可以使用的规则。

首先，许多 MFC 成员函数“能够”在其他线程创建的对象中得到调用。在 MFC 的Include 目录下的 INL 文件中定义的大多数内联函数可以穿过线程界限进行调用，因为它们仅仅是 API 函数的包装。但是调用非内联成员函数就会造成麻烦。例如下列程序不会有问题，它将 CWnd 指针 pWnd 从线程 A 传递给线程 B，而线程 B 通过该指针调用 CWnd::GetParent：

```
CWinThread * pThread = AfxBeginThread (ThreadFunc, pWnd);
   .
   .
   .
UINT ThreadFunc (LPVOID pParam)
{
   CWnd * pWnd = (CWnd * ) pParam;
   CWnd * pParent = pWnd -> GetParent ();
   return 0;
}
```

不过，只是简单地将 GetParent 改为 GetParentFrame 则会导致断言错误：

```
CWinThread * pThread = AfxBeginThread (ThreadFunc, pWnd);
   .
   .
   .
UINT ThreadFunc (LPVOID pParam)
{
   CWnd * pWnd = (CWnd * ) pParam;
   // Get ready for an assertion!
   CWnd * pParent = pWnd -> GetParentFrame ();
   return 0;
}
```

为什么 GetParent 能行而 GetParentFrame 不可以呢？因为 GetParent 几乎直接调用了 API 中的::GetParent 函数。下面给出了在 Afxwin2.inl 中 CWnd::GetParent 的定义，为增强可读性对它做了一点格式修改：

```
AFXWIN_INLINE CWnd * CWnd::GetParent () const
{
   ASSERT (::IsWindow (m_hWnd));
   return CWnd::FromHandle (::GetParent (m_hWnd));
}
```

毫无疑问，m_hWnd 是有效的，它是 pWnd 所指的 CWnd 对象的一部分，并且 FromHandle 将由::GetParent 返回的 HWND 转换为了 CWnd 指针。

但是现在让我们考虑一下当调用 GetParentFrame 时会发生什么情况,其源程序在 Wincore.cpp中可以找到。造成断言错误的一条语句是

```
ASSERT_VALID (this) ;
```

ASSERT_VALID 会调用 CWnd::AssertValid,该函数执行有效的检查,来确保与 this 关联的 HWND 出现在了上结构用来将 HWND 转换为 CWnd 的永久或临时的映射表中。从 CWnd 转换为 HWND 很简单,因为 HWND 是 CWnd 的数据成员,而从 HWND 到 CWnd 的转换就只能通过句柄映射表进行了。并且还有个问题:句柄映射表对于每个线程都是本地使用的,在其他线程中不可见。如果线程 A 创建了 CWnd,它的地址被传递给了 ASSERT_VALID,但是相应的 HWND 却不会在线程 B 的永久或临时的句柄映射表中出现,MFC 就会产生断言错误。许多 MFC 的非内联成员函数要调用 ASSERT_VALID,但是内联函数不会,至少在当前版本中不会。

通常,MFC 的断言处理防止了调用根本不能使用的函数。在发布版本创建时,除了某个线程(在其中创建了父框架)以外的线程调用 GetParentFrame 时,它都会返回 NULL,但是在某些情形下,断言处理错误是假的(就是说,函数无视每个线程的句柄表格内容而照常执行),可以通过传递真实的句柄而非对象指针来避免断言处理发生。例如:如果首先调用 FromHandle 在线程的临时句柄映射表中创建一个输入项,那么就可以安全调用 CWnd::GetTopLevelParent 了,说明如下:

```
CWinThread * pThread = AfxBeginThread (ThreadFunc, pWnd->m_hWnd);
  .
  .
  .
UINT ThreadFunc (LPVOID pParam)
{
  CWnd * pWnd = CWnd::FromHandle ((HWND) pParam);
  CWnd * pParent = pWnd->GetTopLevelParent ();
  return 0;
}
```

这就是 MFC 文献中警告窗口、GDI 对象以及其他对象应该使用句柄而不是指针在线程之间进行传递的原因。通常,如果传递句柄并在目标线程中使用 FromHandle 重新创建对象,就会产生更少的问题。但是别因此而以为任何函数都可以这样。

那么调用属于"纯粹"MFC 类所创建对象的成员函数又会怎样呢?"纯粹"MFC 类,如 CDocument 和 CRect,是指没有封装 HWND、HDC 或其他句柄类型的类,因此它们也不依赖于句柄映射表。答案是:一些可以,一些不可以。下列程序没有问题:

```
CWinThread * pThread = AfxBeginThread (ThreadFunc, pRect);
  .
  .
  .
```