

```

        CFrameWnd::AssertValid();
    }

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

# endif //_DEBUG

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

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
    CCreateContext * pContext)
{
    //
    // Note: Create the CFileView first so the CDriveView's OnInitialUpdate
    // function can call OnUpdate on the CFileView.
    //
    if (!m_wndSplitter.CreateStatic(this, 1, 2) ||
        !m_wndSplitter.CreateView(0, 1, RUNTIME_CLASS(
            CFileView), CSize(0, 0), pContext) ||
        !m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CDriveView),
            CSize(192, 0), pContext))
        return FALSE;

    return TRUE;
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void * pExtra,
    AFX_CMDHANDLERINFO * pHandlerInfo)
{
    //
    // Route to standard command targets first.
    //
    if (CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    //
    // Route to inactive views second.
    //
    CWandererDoc * pDoc = (CWandererDoc *) GetActiveDocument();
    if (pDoc != NULL) { // Important!
        return pDoc->RouteCmdToAllViews(GetActiveView(),
            nID, nCode, pExtra, pHandlerInfo);
    }
    return FALSE;
}

```

WandererDoc.h

```

// WandererDoc.h : interface of the CWandererDoc class
//
////////////////////////////////////

#ifndef AFX_WANDERERDOC_H__AE0A7000_9B0F_11D2_8E53_006008A82731__IN-
CLUDED_
#define AFX_WANDERERDOC_H__AE0A7000_9B0F_11D2_8E53_006008A82731__INCLUD-
ED_

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

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

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWandererDoc)
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:
    BOOL RouteCmdToAllViews (CView * pView, UINT nID, int nCode,
        void * pExtra, AFX_CMDHANDLER_INFO * pHandlerInfo);
    virtual ~CWandererDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CWandererDoc)

```

```

        // 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_WANDERERDOC_H__AE0A7000_9B0F_11D2_8E53_006008A82731 __INCLUDED_)

```

WandererDoc.cpp

```

// WandererDoc.cpp : implementation of the CWandererDoc class
//

# include "stdafx.h"
# include "Wanderer.h"

# include "WandererDoc.h"

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

////////////////////////////////////
// CWandererDoc

IMPLEMENT_DYNCREATE(CWandererDoc, CDocument)

BEGIN_MESSAGE_MAP(CWandererDoc, CDocument)
    //}|AFX_MSG_MAP(CWandererDoc)
    // 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()

////////////////////////////////////
// CWandererDoc construction/destruction

CWandererDoc::CWandererDoc()
{
}

```

```

CWandererDoc::~CWandererDoc()
{
}

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

////////////////////////////////////
// CWandererDoc serialization

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

////////////////////////////////////
// CWandererDoc diagnostics

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

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

////////////////////////////////////
// CWandererDoc commands

BOOL CWandererDoc::RouteCmdToAllViews(CView * pview, UINT nID, int nCode,
    void * pExtra, AFX_CMDHANDLERINFO * pHandlerInfo)
{
    POSITION pos = GetFirstViewPosition();

```

```

        while (pos != NULL) {
            CView* pNextView = GetNextView(pos);
            if (pNextView != pView) {
                if (pNextView->OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
                    return TRUE;
            }
        }
        return FALSE;
    }
}

```

DriveView.h

```

// DriveTreeView.h : interface of the CDriveView class
//
////////////////////////////////////////////////////////////////////

# if !defined(AFX_DRIVEVIEW_H __090B382D_959D_11D2_8F53_006008A82731 __
INCLUDED_)
# define AFX_DRIVEVIEW_H __090B382D_959D_11D2_8F53_006008A82731 __IN
CLUDED_

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

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

// Attributes
public:
    CWandererDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDriveView)
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 ~CDriveView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    BOOL AddDriveItem (LPCWSTR pszDrive);
    int AddDirectories (HTREEITEM hitem, LPCWSTR pszPath);
    void DeleteAllChildren (HTREEITEM hitem);
    void DeleteFirstChild (HTREEITEM hitem);
    CString GetPathFromItem (HTREEITEM hitem);
    BOOL SetButtonState (HTREEITEM hitem, LPCWSTR pszPath);
    int AddDrives ();
    CImageList m_iDrives;
    ///AFX_MSG(CDriveView)
    afx_msg void OnItemExpanding(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnSelectionChanged(NMHDR* pNMHDR, LRESULT* pResult);
    ///AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in DriveTreeView.cpp
inline CWandererDoc* CDriveView::GetDocument()
{ return (CWandererDoc*)m_pDocument; }
#endif
////////////////////////////////////

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

#endif
// !defined(
//     AFX_DRIVEVIEW_H __ 090B382D_959D_11D2_8E53_006008A82731 __ INCLUD-
ED )

```

DriveView.cpp

```

// DriveTreeView.cpp : implementation of the CDriveView class
//

#include "stdafx.h"
#include "Wanderer.h"

```

```

#include "WandererDoc.h"
#include "DriveView.h"

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

// Image indexes
#define ILI_HARDDISK      0
#define ILI_FLOPPY       1
#define ILI_CDROM        2
#define ILI_NET_DRIVE    3
#define ILI_CLOSED_FOLDER 4
#define ILI_OPEN_FOLDER  5

////////////////////////////////////
// CDriveView

IMPLEMENT_DYNCREATE(CDriveView, CTreeView)

BEGIN_MESSAGE_MAP(CDriveView, CTreeView)
    ///AFX_MSG. MAP(CDriveView)
    ON_NOTIFY_REFLECT(TVN_ITEMEXPANDING, OnItemExpanding)
    ON_NOTIFY_REFLECT(TVN_SELCHANGED, OnSelectionChanged)
    ///AFX_MSG. MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDriveView construction/destruction

CDriveView::CDriveView()
{
}

CDriveView::~CDriveView()
{
}

BOOL CDriveView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CTreeView::PreCreateWindow(cs))
        return FALSE;

    cs.style = TVS_HASLINES | TVS_LINESATROOT | TVS_HASBUTTONS |
        TVS_SHOWSELALWAYS;
    return TRUE;
}

```

```

////////////////////////////////////
// CDriveView drawing

void CDriveView::OnDraw(CDC * pDC)
{
    CWandererDoc * pDoc = GetDocument();
    ASSERT_VALID(pDoc);
}

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

    //
    // Initialize the image list.
    //
    m_ilDrives.Create(IDB_DRIVEIMAGES, 16, 1, RGB(255, 0, 255));
    GetTreeCtrl().SetImageList(&m_ilDrives, TVSIL_NORMAL);

    //
    // Populate the tree view with drive items.
    //
    AddDrives();

    //
    // Show the folders on the current drive.
    //
    TCHAR szPath[MAX_PATH];
    ::GetCurrentDirectory(sizeof(szPath)/sizeof(TCHAR), szPath);
    CString strPath = szPath;
    strPath = strPath.Left(3);

    HTREEITEM hItem = GetTreeCtrl().GetNextItem(NULL, TVGN_ROOT);
    while(hItem != NULL) {
        if(GetTreeCtrl().GetItemText(hItem) == strPath)
            break;
        hItem = GetTreeCtrl().GetNextSiblingItem(hItem);
    }

    if(hItem != NULL) {
        GetTreeCtrl().Expand(hItem, TVE_EXPAND);
        GetTreeCtrl().Select(hItem, TVGN_CARET);
    }

    //
    // Initialize the list view.
    //
    strPath = GetPathFromItem(GetTreeCtrl().GetSelectedItem());
    GetDocument() -> UpdateAllViews(this, 0x5A,

```



```

        (CObject *) (LPCTSTR) strPath);
    }

    //////////////////////////////////////
    // CDriveView diagnostics

    #ifdef _DEBUG
    void CDriveView::AssertValid() const
    {
        CTreeView::AssertValid();
    }

    void CDriveView::Dump(CDumpContext& dc) const
    {
        CTreeView::Dump(dc);
    }

    CWandererDoc * CDriveView::GetDocument() // non-debug version is inline
    {
        ASSERT(m_pDocument -> IsKindOf(RUNTIME_CLASS(CWandererDoc)));
        return (CWandererDoc *) m_pDocument;
    }

    #endif // _DEBUG

    //////////////////////////////////////
    // CDriveView message handlers

    int CDriveView::AddDrives()
    {
        int nPos = 0;
        int nDrivesAdded = 0;
        CString string = _T("?\\");

        DWORD dwDriveList = ::GetLogicalDrives();

        while (dwDriveList) {
            if (dwDriveList & 1) {
                string.SetAt(0, _T('A') + nPos);
                if (AddDriveItem(string))
                    nDrivesAdded++;
            }
            dwDriveList >>= 1;
            nPos++;
        }
        return nDrivesAdded;
    }

    BOOL CDriveView::AddDriveItem(LPCTSTR pszDrive)
    {
        CString string;

```

```

HTRHEITEM hItem;

UINT nType = ::GetDriveType(pszDrive);

switch (nType) {

case DRIVE_REMOVABLE:
    hItem = GetTreeCtrl().InsertItem(pszDrive, ILL_FLOPPY,
        ILL_FLOPPY);
    GetTreeCtrl().InsertItem(_T(""), ILL_CLOSED_FOLDER,
        ILL_CLOSED_FOLDER, hItem);
    break;
case DRIVE_FIXED:
case DRIVE_RAMDISK:
    hItem = GetTreeCtrl().InsertItem(pszDrive, ILL_HARD_DISK,
        ILL_HARD_DISK);
    SetButtonState(hItem, pszDrive);
    break;

case DRIVE_REMOTE:
    hItem = GetTreeCtrl().InsertItem(pszDrive, ILL_NET_DRIVE,
        ILL_NET_DRIVE);
    SetButtonState(hItem, pszDrive);
    break;

case DRIVE_CDROM:
    hItem = GetTreeCtrl().InsertItem(pszDrive, ILL_CD_ROM,
        ILL_CD_ROM);
    GetTreeCtrl().InsertItem(_T(""), ILL_CLOSED_FOLDER,
        ILL_CLOSED_FOLDER, hItem);
    break;

default:
    return FALSE;
}

return TRUE;
}

BOOL CDriveView::SetButtonState(HTRHEITEM hItem, LPCTSTR pszPath)
{
    HANDLE hFind;
    WIN32_FIND_DATA fd;
    BOOL bResult = FALSE;

    CString strPath = pszPath;
    if (strPath.Right(1) != _T("."))
        strPath += _T(".");
    strPath += _T("*.");

```

```

    if ((hFind = ::FindFirstFile(strPath, &fd)) == INVALID_HANDLE_VALUE)
        return bResult;

    do {
        if (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            CString strComp = (LPCTSTR) &fd.cFileName;
            if ((strComp != _T(".")) && (strComp != _T(".."))) {
                GetTreeCtrl().InsertItem(_T(""), ILI_CLOSED_FOLDER,
                    ILI_CLOSED_FOLDER, hitem);
                bResult = TRUE;
                break;
            }
        }
    } while (::FindNextFile(hFind, &fd));

    ::FindClose(hFind);
    return bResult;
}

void CDriveView::OnItemExpanding(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
    HTREEITEM hItem = pNMTreeView->itemNew.hitem;
    CString string = GetPathFromItem(hItem);

    *pResult = FALSE;

    if (pNMTreeView->action == TVE_EXPAND) {
        DeleteFirstChild(hItem);
        if (AddDirectories(hItem, string) == C)
            *pResult = TRUE;
    }
    else { // pNMTreeView->action == TVE_COLLAPSE
        DeleteAllChildren(hItem);
        if (GetTreeCtrl().GetParentItem(hItem) == NULL)
            GetTreeCtrl().InsertItem(_T(""), ILI_CLOSED_FOLDER,
                ILI_CLOSED_FOLDER, hitem);
        else
            SetButtonState(hItem, string);
    }
}

CString CDriveView::GetPathFromItem(HTREEITEM hItem)
{
    CString strResult = GetTreeCtrl().GetItemText(hItem);

    HTREEITEM hParent;
    while ((hParent = GetTreeCtrl().GetParentItem(hItem)) != NULL) {
        CString string = GetTreeCtrl().GetItemText(hParent);

```

```

        if (string.Right (1) != _T ("\\"))
            string += _T ("\\"");
        strResult = string + strResult;
        hItem = hParent;
    }
    return strResult;
}

void CDriveView::DeleteFirstChild(HTREEITEM hItem)
{
    HTREEITEM hChildItem;
    if ((hChildItem = GetTreeCtrl ().GetChildItem (hItem)) != NULL)
        GetTreeCtrl ().DeleteItem (hChildItem);
}

void CDriveView::DeleteAllChildren(HTREEITEM hItem)
{
    HTREEITEM hChildItem;
    if ((hChildItem = GetTreeCtrl ().GetChildItem (hItem)) == NULL)
        return;

    do {
        HTREEITEM hNextItem =
            GetTreeCtrl ().GetNextSiblingItem (hChildItem);
        GetTreeCtrl ().DeleteItem (hChildItem);
        hChildItem = hNextItem;
        while (hChildItem != NULL);
    }

}

int CDriveView::AddDirectories(HTREEITEM hItem, LPCTSTR pszPath)
{
    HANDLE hFind;
    WIN32_FIND_DATA fd;
    HTREEITEM hNewItem;

    int nCount = 0;

    CString strPath = pszPath;
    if (strPath.Right (1) != _T ("\\"))
        strPath += _T ("\\"");
    strPath += _T ("*.*");

    if ((hFind = ::FindFirstFile (strPath, &fd)) == INVALID_HANDLE_VALUE) {
        if (GetTreeCtrl ().GetParentItem (hItem) == NULL)
            GetTreeCtrl ().InsertItem (_T (""), ILI_CLOSED_FOLDER,
                ILI_CLOSED_FOLDER, hItem);
        return 0;
    }
    do {

```

```

        if (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            CString strComp = (LPCTSTR) &fd.cFileName;
            if ((strComp != _T(".")) && (strComp != _T(".."))) {
                hNewItem =
                    GetTreeCtrl().InsertItem((LPCTSTR) &fd.cFileName,
                    ILI_CLOSED_FOLDER, ILI_OPEN_FOLDER, hItem);

                CString strNewPath = pszPath;
                if (strNewPath.Right(1) != _T("\\"))
                    strNewPath += _T("\");

                strNewPath += (LPCTSTR) &fd.cFileName;
                SetButtonState(hNewItem, strNewPath);
                nCount++;
            }
        }
    } while (::FindNextFile(hFind, &fd));

    ::FindClose(hFind);
    return nCount;
}

void CDriveView::OnSelectionChanged(NMHDR * pNMHDR, LRESULT * pResult)
{
    NM_TREEVIEW * pNMTreeView = (NM_TREEVIEW *) pNMHDR;
    CString strPath = GetPathFromItem(pNMTreeView->itemNew.hItem);
    GetDocument() -> UpdateAllViews(this, 0x5A,
        (CObject *) (LPCTSTR) strPath);
    *pResult = 0;
}

```

FileView.h

```

// FileView.h : interface of the CFileView class
//
///////////////////////////////////////////////////////////////////
# if !defined(AFX_FILEVIEW_H __18BD7B80_95C6_11D2_8E53_006008A82731 __IN-
CLUDED_)
# define AFX_FILEVIEW_H __18BD7B80_95C6_11D2_8E53_006008A82731 __INCLUDED_

# if _MSC_VER > 1000
# pragma once
# endif // _MSC_VER > 1000
typedef struct tagITEMINFO {
    CString      strFileName;
    DWORD       nFileSizeLow;
    FILETIME     ftLastWriteTime;
}

```

```

    } ITEMINFO;

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

// Attributes
public:
    CWandererDoc * GetDocument();

// Operations
public:
    static int CALLBACK CompareFunc (LPARAM lParam1, LPARAM lParam2,
        LPARAM lParamSort);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFileView)
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
    virtual void OnUpdate(CView * pSender, LPARAM lHint, CObject * pHint);
    //{{AFX_VIRTUAL

// Implementation
public:
    int Refresh (LPCTSTR pszPath);
    virtual ~CFileView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Generated message map functions
protected:
    CString m_strPath;
    void FreeItemMemory ();
    BOOL AddItem (int nIndex, WIN32_FIND_DATA * pfd);
    CImageList m_ilSmall;
    CImageList m_ilLarge;
    //{{AFX_MSG(CFileView)
    afx_msg void OnDestroy();

```

```

    afx_msg void OnGetDispInfo(NMHDR * pNMHDR, LRESULT * pResult);
    afx_msg void OnColumnClick(NMHDR * pNMHDR, LRESULT * pResult);
    afx_msg void OnViewLargeIcons();
    afx_msg void OnViewSmallIcons();
    afx_msg void OnViewList();
    afx_msg void OnViewDetails();
    afx_msg void OnUpdateViewLargeIcons(CCmdUI * pCmdUI);
    afx_msg void OnUpdateViewSmallIcons(CCmdUI * pCmdUI);
    afx_msg void OnUpdateViewList(CCmdUI * pCmdUI);
    afx_msg void OnUpdateViewDetails(CCmdUI * pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in FileView.cpp
inline CWandererDoc * CFileView::GetDocument()
    { return (CWandererDoc *)m_pDocument; }
#endif

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

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

#endif
// !defined(AFX_FILEVIEW_H __ 18BD7B80_95C6_11D2_8E53_006008A82731 __ INCLUD-
ED_)

```

FileView.cpp

```

// FileView.cpp : implementation of the CFileView class
//

#include "stdafx.h"
#include "Wanderer.h"
#include "WandererDoc.h"
#include "FileView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

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

// CFileView

IMPLEMENT_DYNCREATE(CFileView, CListView)

```

```

BEGIN_MESSAGE_MAP(CFileView, CListView)
    ///AFX_MSG_MAP(CFileView)
    ON_WM_DESTROY()
    ON_NOTIFY_REFLECT(LVN_GETDISPINFO, OnGetDispInfo)
    ON_NOTIFY_REFLECT(LVN_COLUMNCLICK, OnColumnClick)
    ON_COMMAND(ID_VIEW_LARGE_ICONS, OnViewLargeIcons)
    ON_COMMAND(ID_VIEW_SMALL_ICONS, OnViewSmallIcons)
    ON_COMMAND(ID_VIEW_LIST, OnViewList)
    ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
    ON_UPDATE_COMMAND_UI(ID_VIEW_LARGE_ICONS, OnUpdateViewLargeIcons)
    ON_UPDATE_COMMAND_UI(ID_VIEW_SMALL_ICONS, OnUpdateViewSmallIcons)
    ON_UPDATE_COMMAND_UI(ID_VIEW_LIST, OnUpdateViewList)
    ON_UPDATE_COMMAND_UI(ID_VIEW_DETAILS, OnUpdateViewDetails)
    ///AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CFileView construction/destruction

CFileView::CFileView()
{
}

CFileView::~CFileView()
{
}

BOOL CFileView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CListView::PreCreateWindow(cs))
        return FALSE;

    cs.style &= ~LVS_TYPEMASK;
    cs.style |= LVS_REPORT;
    return TRUE;
}

////////////////////////////////////
// CFileView drawing

void CFileView::OnDraw(CDC* pDC)
{
    CWandererDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
}

void CFileView::OnInitialUpdate()
{
}

```



```

int CFileView::Refresh(LPCTSTR pszPath)
{
    CString strPath = pszPath;
    if (strPath.Right(1) != _T("\\"))
        strPath += _T("\\");
    strPath += _T("*.");

    HANDLE hFind;
    WIN32_FIND_DATA fd;
    int nCount = 0;

    if ((hFind = ::FindFirstFile(strPath, &fd)) != INVALID_HANDLE_VALUE) {
        //
        // Delete existing items (if any).
        //
        GetListCtrl().DeleteAllItems();

        //
        // Show the path name in the frame window's title bar.
        //
        TCHAR szFullPath[MAX_PATH];
        ::GetFullPathName(pszPath, sizeof(szFullPath) / sizeof(TCHAR),
            szFullPath, NULL);
        m_strPath = szFullPath;

        CString strTitle = _T("WinDir - ");
        strTitle += szFullPath;
        AfxGetMainWnd() -> SetWindowText(strTitle);

        //
        // Add items representing files to the list view.
        //
        if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
            AddItem(nCount++, &fd);

        while (::FindNextFile(hFind, &fd)) {
            if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
                if (!AddItem(nCount++, &fd))
                    break;
        }
        ::FindClose(hFind);
    }
    return nCount;
}

BOOL CFileView::AddItem(int nIndex, WIN32_FIND_DATA * pfd)
{
    //
    // Allocate a new ITEMINFO structure and initialize it with information

```

```

        // about the item.
        //
        ITEMINFO * pItem;
        try {
            pItem = new ITEMINFO;
        }
        catch (CMemoryException * e) {
            e->Delete();
            return FALSE;
        }

        pItem->strFileName = pfd->cFileName;
        pItem->nFileSizeLow = pfd->nFileSizeLow;
        pItem->ftLastWriteTime = pfd->ftLastWriteTime;

        //
        // Add the item to the list view.
        //
        LV_ITEM lvi;
        lvi.mask = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM;
        lvi.iItem = nIndex;
        lvi.iSubItem = 0;
        lvi.iImage = 0;
        lvi.pszText = LPSTR_TEXTCALLBACK;
        lvi.lParam = (LPARAM) pItem;

        if (GetListCtrl().InsertItem(&lvi) == -1)
            return FALSE;

        return TRUE;
    }

void CFileView::FreeItemMemory()
{
    int nCount = GetListCtrl().GetItemCount();
    if (nCount) {
        for (int i = 0; i < nCount; i++)
            delete (ITEMINFO *) GetListCtrl().GetItemData(i);
    }
}

void CFileView::OnDestroy()
{
    FreeItemMemory();
    CListView::OnDestroy();
}

void CFileView::OnGetDispInfo(NMHDR * pNMHDR, LRESULT * pResult)
{

```

```

CString string;
LV_DISPINFO* pDispInfo = (LV_DISPINFO*) pNMHDR;

if (pDispInfo->item.mask & LV_F_TEXT) {
    ITEMINFO* pItem = (ITEMINFO*) pDispInfo->item.lParam;

    switch (pDispInfo->item.iSubItem) {

    case 0: // File name
        ::lstrcpy (pDispInfo->item.pszText, pItem->strFileName);
        break;

    case 1: // File size
        string.Format (_T ("%u"), pItem->nFileSizeLow);
        ::lstrcpy (pDispInfo->item.pszText, string);
        break;

    case 2: // Date and time
        CTime time (pItem->ftLastWriteTime);

        . BOOL pm = FALSE;
        int nHour = time.GetHour ();
        if (nHour == 0)
            nHour = 12;
        else if (nHour == 12)
            pm = TRUE;
        else if (nHour > 12) {
            nHour -= 12;
            pm = TRUE;
        }

        string.Format (_T ("%d/%0.2d/%0.2d (%d:%0.2d%c)"),
            time.GetMonth (), time.GetDay (), time.GetYear () % 100,
            nHour, time.GetMinute (), pm ? _T ('p') : _T ('a'));
        ::lstrcpy (pDispInfo->item.pszText, string);
        break;

    }

    pResult = 0;
}

void CFileView::OnColumnClick(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*) pNMHDR;
    GetListCtrl().SortItems (CompareFunc, pNMListView->iSubItem);
    *pResult = 0;
}

int CALLBACK CFileView::CompareFunc (LPARAM lParam1, LPARAM lParam2,
    LPARAM lParamSort)

```

```

    {
        ITEMINFO * pItem1 = (ITEMINFO *) lParam1;
        ITEMINFO * pItem2 = (ITEMINFO *) lParam2;
        int nResult;

        switch (lParamSort) {

        case 0: // File name
            nResult = pItem1->strFileName.CompareNoCase(pItem2->strFileName);
            break;

        case 1: // File size
            nResult = pItem1->nFileSizeLow - pItem2->nFileSizeLow;
            break;

        case 2: // Date and time
            nResult = ::CompareFileTime(&pItem1->ftLastWriteTime,
                &pItem2->ftLastWriteTime);
            break;
        }

        return nResult;
    }

void CFileView::OnViewLargeIcons()
{
    ModifyStyle(LVS_TYPEMASK, LVS_ICON);
}

void CFileView::OnViewSmallIcons()
{
    ModifyStyle(LVS_TYPEMASK, LVS_SMALLICON);
}

void CFileView::OnViewList()
{
    ModifyStyle(LVS_TYPEMASK, LVS_LIST);
}

void CFileView::OnViewDetails()
{
    ModifyStyle(LVS_TYPEMASK, LVS_REPORT);
}

void CFileView::OnUpdateViewLargeIcons(CCmdUI * pCmdUI)
{
    DWORD dwCurrentStyle = GetStyle() & LVS_TYPEMASK;
    pCmdUI->SetRadio(dwCurrentStyle == LVS_ICON);
}

void CFileView::OnUpdateViewSmallIcons(CCmdUI * pCmdUI)

```

```

    |
    |     DWORD dwCurrentStyle = GetStyle () & LVS_TYPEMASK;
    |     pCmdUI->SetRadio (dwCurrentStyle == LVS_SMALLICON);
    |
    |
void CFileView::OnUpdateViewList(CCmdUI * pCmdUI)
    |
    |     DWORD dwCurrentStyle = GetStyle () & LVS_TYPEMASK;
    |     pCmdUI->SetRadio (dwCurrentStyle == LVS_LIST);
    |
    |
void CFileView::OnUpdateViewDetails(CCmdUI * pCmdUI)
    |
    |     DWORD dwCurrentStyle = GetStyle () & LVS_TYPEMASK;
    |     pCmdUI->SetRadio (dwCurrentStyle == LVS_REPORT);
    |
    |
void CFileView::OnUpdate(CView * pSender, LPARAM lHint, CObject * pHint)
    |
    |     if (lHint == 0x5A) {
    |         FreeItemMemory ();
    |         GetListCtrl ().DeleteAllItems ();
    |         Refresh ((LPCTSTR) pHint);
    |         return;
    |     }
    |     CListView::OnUpdate (pSender, lHint, pHint);
    |

```

图 11-8 Wanderer 应用程序

在创建 Wanderer 的过程中,使用了 AppWizard 生成的 SDI 文档/视图应用程序源程序代码,插入了第 10 章中讲述的 CDriveView 和 CFileView 类并按上面的讲述对其进行了修改,给 CMainFrame 添加了 CSplitterWnd 成员变量,覆盖了 OnCreateClient,并加入了对 CreateStatic 和 CreateView 的调用。但是还有一种方法可以创建资源管理器类型的应用程序。在 AppWizard 的 Step 5 对话框中如果选择了 Windows Explorer 而不是 MFC Standard,AppWizard 就会添加代码来创建静态拆分窗口。它也派生了一对视图类:一个从 CTreeView,另一个从 CListView 或您选择的视图类中,并将它们放在拆分窗口的窗格内。不幸的是 AppWizard 生成的视图类并没有给派生它们的基类添加多少东西,您还得亲自编写相当多的程序来创建资源管理器类型的应用程序。

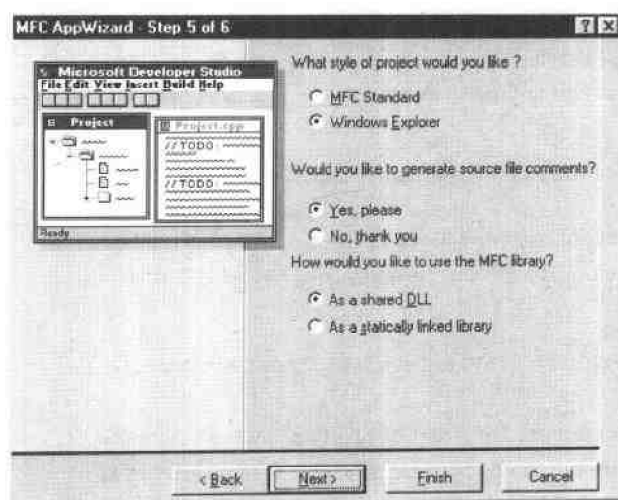


图 11-9 使用 AppWizard 创建资源管理器样式的应用程序

尽管 Wanderer 和 Windows 资源管理器外表相似,但这两个应用程序的根本差别使得它们的特性大相径庭。Wanderer 是“文件浏览器”,它显示驱动器、目录以及文件名称。资源管理器则是一个“名字空间浏览器”,它被作为命令解释器名字空间的可视窗口。研究一下资源管理器窗口的左窗格,您可以看到命令解释器的名字空间的构造以及所包含的对象种类。桌面对象位于分层结构的最顶层,下面是我的电脑、网上邻居、回收站,下一层是驱动器,等等。驱动器、目录、文件仅仅是命令解释器中名字空间的子集。名字空间还包括打印机、打印机文件夹以及其他在文件系统中没有直接类似物的对象。操作系统的命令解释器支持一组自身的 API 函数,应用程序可以用来访问它的名字空间。一些是常规 API 函数如 `SHGetDesktopFolder` 等,其他是 COM 函数,要通过 `IShellFolder` 接口访问。想了解更多信息,可在 MSDN 中查找有关命令解释器名字空间的文献。

11.2.5 自定义命令传送

正如您所知道的,MFC 的 `CFrameWnd` 类将其接收到的命令消息和用户界面(UI)更新消息传递给了其他的对象,使得框架窗口不必处理菜单项和其他 UI 对象命令。多亏有了命令传送机制,使得在应用程序对象、文档对象或视图对象中对涉及菜单项和工具栏按钮的处理就像在框架窗口类中一样容易了。第 9 章中介绍了命令传送机制,图 9-2 说明了在 SDI 框架窗口接收到命令或 UI 更新消息之后消息所经过的路径。活动视图首先接收到消息,然后是文档、文档模板、框架窗口,最后是应用程序对象。对于大多数文档/视图应用程序,图 9-2 描述的命令传送顺序就能满足要求了,它给每个可能要接收命令或更新消息的对象都提供了处理消息的机会。

偶尔您也会遇到这样的应用程序,其默认的命令传送机制不能满足要求。Wanderer 就是这样的一个例子,为什么呢? Wanderer 的 View 菜单中针对视图项目的命令和更新要在 CFileView 类中处理。如果 CFileView 是活动视图的话,其命令和更新处理程序工作正常,因为在主结构的传送列表中包含有活动视图。但是当 CDriveView 是活动视图时,由于 CFileView 不是活动视图,所以它不会被通知有关 View 中命令的事件。因此,在左窗格中的 CDriveView 具有输入焦点时,Options 菜单中的命令就会被灰化不能被选中。

为解决此问题,Wanderer 修改了命令传送顺序,使得不会被任何标准命令目标对象处理的命令和更新消息被传送给不活动视图。在 CMainFrame::OnCmdMsg 中进行了处理,首先调用 CFrameWnd::OnCmdMsg 将命令和更新消息提交给标准命令目标对象:

```
if (CFrameWnd::OnCmdMsg (nID, nCode, pExtra, pHandlerInfo))
    return TRUE;
```

如果 CFrameWnd::OnCmdMsg 返回 0,表明没有标准命令目标对象处理消息,CMainFrame::OnCmdMsg 就调用文档类中的函数将消息传送给所有不活动的视图:

```
CWandererDoc * pDoc = (CWandererDoc *) GetActiveDocument ();
if (pDoc != NULL) { // Important!
    return pDoc->RouteCmdToAllViews (GetActiveView (),
        nID, nCode, pExtra, pHandlerInfo);
}
```

CWandererDoc::RouteCmdToAllViews 枚举每个与文档关联的视图,并调用每个视图的 OnCmdMsg 函数:

```
BOOL CWandererDoc::RouteCmdToAllViews(CView *pView, UINT nID, int nCode,
    void *pExtra, AFX_CMDHANDLERINFO pHandlerInfo)
{
    POSITION pcs = GetFirstViewPosition();

    while (pos != NULL) {
        CView * pNextView = GetNextView(pos);
        if (pNextView != pView)
            if (pNextView->OnCmdMsg (nID, nCode, pExtra, pHandlerInfo))
                return TRUE;
    }

    return FALSE;
}
```

CMainFrame::OnCmdMsg 传递给 RouteCmdToAllViews 一个指向活动视图的指针,以便 RouteCmdToAllViews 避免调用活动视图的 OnCmdMsg 函数。活动视图作为标准命令传送顺序中的一部分已经被调用过了,再调用它是一种浪费。框架窗口提供指向活动视图的指针,

这是因为文档类区分不出活动与不活动视图。出于同样原因,框架窗口知道哪个是活动视图,但不知道到底有多少视图存在。这就是 CMainFrame 要调用文档类中的函数来枚举视图而不是靠自己来枚举的原因。

注意在一些 MFC 版本中由 GetNextView 返回的 CView 指针必须被强制转换为 CCmdTarget 指针,因为在那些版本中错误地将 OnCmdMsg 在 CView 中声明为了保护型。我们由衷地感谢,这个问题在 MFC 6.0 中被解决了。

自定义传送是给非标准命令目标对象传送命令和 UI 更新消息的功能强大的工具。只要覆盖合适的 OnCmdMsg 函数,您就可以在命令传送顺序中的任何地方进行处理。一般情况下,应该从覆盖函数中调用基类中的 OnCmdMsg 以确保默认命令传送继续有效。要小心处理所调用的 OnCmdMsg 函数,如果调用不当可能会陷入递归循环,就是对象 A 调用对象 B,对象 B 再调用对象 A。例如:不要从文档的 OnCmdMsg 函数中调用视图的 OnCmdMsg 函数,因为视图会将文档作为标准命令传送顺序的一部分来调用。

11.2.6 嵌套拆分窗口

通过嵌套静态拆分窗口可以创建与 Microsoft Outlook Express 具有的拆分窗口相似的嵌套拆分窗口。下列 OnCreateClient 函数创建了一个嵌套静态拆分窗口,分为两列。右列又进一步分为两行。用户可以通过拖动拆分条来调整窗格的相对大小,但由于拆分窗口是静态而非动态的,所以基本外观不会被改变:

```
BOOL CMainFrame::OnCreateClient (LPCREATESTRUCT lpCreateStruct,
    CCreateContext * pContext)
{
    if (!m_wndSplitter1.CreateStatic (this, 1, 2) ||
        !m_wndSplitter1.CreateView (0, 0, RUNTIME_CLASS (CTextView),
            CSize (128, 0), pContext) ||
        !m_wndSplitter2.CreateStatic (&m_wndSplitter1, 2, 1, WS_CHILD |
            WS_VISIBLE, m_wndSplitter1.IdFromRowCol (0, 1)) ||
        !m_wndSplitter2.CreateView (0, 0, RUNTIME_CLASS (CPictureView),
            CSize (0, 128), pContext) ||
        !m_wndSplitter2.CreateView (1, 0, RUNTIME_CLASS (CPictureView),
            CSize (0, 0), pContext))
        return FALSE;
    return TRUE;
}
```

以下简要说明了 if 语句中创建和初始化嵌套拆分窗口的过程:

1. 第一个拆分窗口是根据 CSplitterWnd 的数据成员 m_wndSplitter1 调用 CreateStatic 而创建的。此拆分窗口包含一行两列。
2. 使用 CreateView 将 CTextView 添加到 m_wndSplitter1 的第一个(左边)窗格中。

3. 通过调用 `m_wndSplitter2` 的 `CreateStatic` 函数在第一个拆分窗口的右窗格中创建了第二个拆分窗口。`m_wndSplitter2` 的父亲是 `m_wndSplitter1` 而不是框架窗口,并且它得到一个子窗口 ID,将其标识为第 0 行第 1 列的窗格。调用 `CSplitterWnd::IdFromRowCol` 可以得到 `m_wndSplitter2` 适当的 ID,该函数使用简单的换算将行列编号转换为添加给 `AFX_IDW_PANE_FIRST` 的数字偏移量。
4. 调用 `CreateView` 两次给每个 `m_wndSplitter2` 窗格添加 `CPictureView`。

由于 MFC 对给动态创建的拆分窗口窗格生成要填入的新视图作了一些假定,所以对 `m_wndSplitter2` 使用动态拆分窗口要稍微复杂一些。如果您试图给一个静态拆分窗口嵌套动态拆分窗口,如下:

```

BOOL CMainFrame::OnCreateClient (LPCREATESTRUCT lpCreateStruct,
    CCreateContext * pContext)
{
    if (! m_wndSplitter1.CreateStatic (this, 1, 2) ||
        ! m_wndSplitter1.CreateView (0, 0, RUNTIME_CLASS (CTextView),
            CSize (128, 0), pContext) ||
        ! m_wndSplitter2.Create (&m_wndSplitter1, 2, 1, CSize (1, 1),
            pContext, WS_CHILD | WS_VISIBLE | WS_HSCROLL |
            WS_VSCROLL | SPLS_DYNAMIC_SPLIT,
            m_wndSplitter1.IdFromRowCol (0, 1)))
        return FALSE;
    return TRUE;
}

```

有时在拆分动态拆分窗口时会产生访问错误。其中原因根植于主结构中。当动态拆分窗口拆分时, `CSplitterWnd` 用一个 `NULL` `pContext` 指针来调用 `CreateView` 为新窗格创建一个新视图。由于 `pContext` 为 `NULL` 值, `CreateView` 就要向框架窗口查询指向活动视图的指针,并使用该视图作为新视图的模型。如果碰巧在拆分时 `CTextView` 窗口是活动视图,主结构看到该视图不是动态拆分窗口的子视图就会创建一个与文档对象无关的“空”视图。这样在视图第一次访问其文档时,访问错误就会发生。

成功地在静态拆分窗口中嵌套动态拆分窗口的秘密有两条:

1. 从 `CSplitterWnd` 中派生类并用以下程序代码替换派生类中的 `CSplitterWnd::SplitRow`:

```

BOOL CNestedSplitterWnd::SplitRow (int cyBefore)
{
    GetParentFrame () ->
        SetActiveView ((CView*) GetPane (0, 0));
    return CSplitterWnd::SplitRow (cyBefore);
}

```

2. 使要嵌套的动态拆分窗口成为派生类的实例而不是 `CSplitterWnd` 的实例。
`SplitRow` 是一个虚拟 `CSplitterWnd` 函数,在拖动水平拆分条创建新窗格时要调用它。以

上给出的 SplitRow 函数的实现使得动态拆分窗口内最顶层窗格中的视图在拆分进行之前成为了活动视图,这样就漂亮地解决了由于活动视图是静态拆分窗口的子视图而造成的动态视图创建中的问题。覆盖函数使用了 GetParentFrame 而不是 GetParent,这是因为动态拆分窗口的父亲实际上是静态拆分窗口而不是框架窗口,而且是框架窗口函数(不是静态拆分窗口函数)设置活动视图的。

11.2.7 带有多种视图类型的动态拆分窗口

上一节讲述了一种通过从 CSplitterWnd 派生类并覆盖 CSplitterWnd::SplitRow 来自定义拆分窗口的方法。CSplitterWnd 类中还包含其他可以覆盖来自定义拆分窗口功能的虚拟函数。其中一个是 CreateView 函数,MFC 在动态拆分窗口被拆分时调用来创建新视图。可以用以下方法创建动态拆分窗口,在它的不同窗格中显示不同类型的视图:从 CSplitterWnd 中派生一个类,并覆盖 CreateView,然后用指向所选视图的 CRuntimeClass 指针调用 CSplitterWnd::CreateView。

下列 CreateView 覆盖函数不管第 0 行第 0 列中的视图类型是什么,都将迫使第 1 行第 0 列的窗格具有 CTextView 视图:

```
BOCL CDynaSplitterWnd::CreateView(int row, int col,
    CRuntimeClass * pViewClass, SIZE sizeInit,
    CCreateContext * pContext)
{
    if ((row == 1) && (col == 0))
        return CSplitterWnd::CreateView(row, col,
            RUNTIME_CLASS(CTextView), sizeInit, pContext);

    return CSplitterWnd::CreateView(row, col, pViewClass,
        sizeInit, pContext);
}
```

对于您所使用的每种不同的拆分窗口,您可能必须修改此程序代码,因为其中视图类与行列编号被硬性联系在了一起。但是您可以创建一个一般的(可重用的)动态拆分窗口类,它通过添加一个 RegisterView 函数将由 CRuntimeClass 指针标识的视图类型与行列编号相联系,从而实现了多视图类型的支持。在调用 CSplitterWnd::Create 之前,拆分窗口要用有关每个窗格中视图类型的信息进行初始化,然后 CreateView 就会使用此信息生成相应的视图了。

第 12 章 工具栏、状态栏和组合栏

这一章,我们将介绍两个新类:CToolBar 和 CStatusBar。您可以运用这两个类进一步完善应用程序的用户界面。CToolBar 可以实现工具栏。工具栏是包含命令按钮(有时还包含其他类型的控件)的带状窗口。通过这些按钮可以方便、快捷地访问常用命令。CStatusBar 是 MFC 的状态栏类。状态栏是一个帮助窗口,它可以显示菜单项和工具栏按钮的掠过式帮助信息,以及其他帮助信息。在 MFC 应用程序中添加工具栏和状态栏非常容易,因为 CToolBar 和 CStatusBar 全面封装了这些常用用户界面(UI)元素。

本章还会讲到 MFC 的另一个类 CReBar,该类封装了 Microsoft Internet Explorer 引入的组合栏控件。组合栏将普通工具栏转换为 Internet Explorer、Microsoft Visual C++ 和其他 Microsoft 应用程序中特有的格式化工具栏(也称为“coolbars”)。它们也是“命令栏”的基础。命令栏是带有菜单项的菜单栏,当光标经过时各菜单项显示成按钮。看看 Visual C++ 中作为主菜单的命令栏,您就会明白。有了 CReBar,MFC 编程人员只需用一两行代码就可以将 CToolBar 转换为格式化工具栏。很快您就会看到这个转换过程。

12.1 工具栏

制作工具栏的目的是:通过单击就能访问常用命令。工具栏按钮实际上成了菜单命令的快捷方式,但是它们也实现菜单中没有的命令。MFC 的 CToolBar 类在取得位图资源(其中包含工具栏按钮上的图像和按钮 ID 数组)之后,创建一个工具栏对象,或停放在框架窗口的一侧,或浮动在自己的小框架窗口中。和菜单项一样,也要赋给工具栏按钮命令 ID。单击工具栏按钮和选中菜单项都能产生 WM_COMMAND 消息。如果某菜单项和工具栏按钮的命令 ID 相同,则一个命令处理程序可以用于两种操作方式。只需几行语句就可以给工具栏添加组合框、复选框和其他控件;还可以将普通按钮转换为“复选按钮”(单击时)或“单选按钮”(和单选按钮工作方式相似)。MFC 提供隐藏和显示工具栏、保存和恢复工具栏的函数,还有其他很多函数。

在早期版本的 MFC 中,CToolBar 是独立的类,它的功能完全来自 MFC。如今,CToolBar 从 Comet32.dll 的工具栏控件得到了许多功能。另一个但较原始的 MFC 类 CToolBarCtrl 提供了工具栏控件的 MFC 接口。了解这一点是必要的。因为如果您想用 CToolBar 做些工作,但又找不到合适的成员函数,CToolBarCtrl 可能会有您需要的成员函数。如果您首先调用 CToolBar::GetToolBarCtrl 得到该控件的 CToolBarCtrl 引用,您就能对 CToolBar 调用 CToolBarCtrl 函数了。然

而,大多数时候,CToolBar 开始会做您要求的所有工作,随后便只做其中一部分了。了解这一点后,我们再看看建立 CToolBar 并让它运行起来到底需要哪些工作。

12.1.1 创建和初始化工具栏

通过构造 CToolBar 对象并调用 CToolBar::Create 可以创建工具栏。因为工具栏是应用程序主框架窗口的子窗口,通常随框架窗口的创立而创立,所以一般在框架窗口类中添加一个 CToolBar 成员,并在框架窗口的 OnCreate 处理程序中调用 Create。如果 m_wndToolBar 是一个 CToolBar 数据成员,则语句

```
m_wndToolBar.Create(this);
```

创建一个工具栏,它是 this 的子窗口。有两个参数在调用中是隐含的:工具栏的样式和它的子窗口 ID。默认样式为 WS_CHILD|WS_VISIBLE|CBRS_TOP。通过给 Create 再添加一个参数,或在工具栏创建后调用 SetBarStyle 函数(从它的基类 CControlBar 继承而来),可改变工具栏的样式。例如:如果要用样式 CBRS_BOTTOM 取代 CBRS_TOP,使工具栏和父窗口的底部对齐,可以这样创建它:

```
m_wndToolBar.Create(this, WS_CHILD|WS_VISIBLE|CBRS_BOTTOM);
```

也可以这样创建它:

```
m_wndToolBar.Create(this);
m_wndToolBar.SetBarStyle((m_wndToolBar.GetBarStyle() &
~CBRS_TOP) | CBRS_BOTTOM);
```

CToolBar::Create 的第 3 个可选参数用于指定工具栏 ID,其默认值为 AFX_IDW_TOOLBAR。除非编写包含两个或更多个工具栏的应用程序,一般不必改变该工具栏 ID 默认值。在有多 个工具栏的应用程序中,要给每个工具栏不同的 ID。

新建的工具栏是空的,因此下一步工作是在其中添加按钮。添加按钮的一种方法是:调用 CToolBar::LoadBitmap 加载包含按钮表面图像的位图资源,并调用 CToolBar::SetButtons 通知工具栏有多少个按钮,以及这些按钮的命令 ID 是什么。下列语句创建一个工具栏,并用位图资源 IDR_TOOLBAR 中的图像和 nButtonIDs 数组中的 ID 将它初始化。ID_SEPARATOR 值则在按钮间插入几个像素宽的间距。

```
// In the RC file
IDR_TOOLBAR BITMAP Toolbar.bmp

// In the CPP file
static UINT nButtonIDs[] = {
    ID_FILE_NEW,
    ID_FILE_OPEN,
```

```

        ID_FILE_SAVE,
        ID_SEPARATOR,
        ID_EDIT_CUT,
        ID_EDIT_COPY,
        ID_EDIT_PASTE,
        ID_EDIT_UNDO,
        ID_SEPARATOR,
        ID_FILE_PRINT
    };

    m_wndToolBar.Create(this);
    m_wndToolBar.LoadBitmap(IDR_TOOLBAR);
    m_wndToolBar.SetButtons(nButtonIDs, 10);

```

位图资源包含所有工具栏按钮图像,它们和电影胶片中的帧一样从一端排到另一端,如图 12-1 所示。默认方式下,每个图像都是 16 个像素宽、15 个像素高。按钮本身是 24 个像素宽、22 个像素高。

工具栏位图



结果工具栏



图 12-1 工具栏图像和由该图像生成的工具栏

用 `CToolBar::SetSizes` 可以改变图像和按钮尺寸。要设计专业水平的工具栏按钮则需要一点艺术感,但对于标准按钮,比如 New、Open、Save、Cut、Copy、Paste 和 Print,从 Visual C++ 提供的 `Toolbar.bmp` 位图中借用图像就行了。

创建工具栏按钮的第二种方法是:在应用程序的 RC 文件中添加一个描述按钮 ID 和图像尺寸的 `TOOLBAR` 资源,并通过该资源 ID 调用 `CToolBar::LoadToolBar`。下列语句创建了一个工具栏,并将它初始化。该工具栏和前段中的那个一样:

```

// In the RC file
IDR_TOOLBAR BITMAP Toolbar.bmp

IDR_TOOLBAR TOOLBAR 16, 15
BEGIN
    BUTTON ID_FILE_NEW

```

```

        BUTTON ID_FILE_OPEN
        BUTTON ID_FILE_SAVE
        SEPARATOR
        BUTTON ID_EDIT_CUT
        BUTTON ID_EDIT_COPY
        BUTTON ID_EDIT_PASTE
        BUTTON ID_EDIT_UNDO
        SEPARATOR
        BUTTON ID_FILE_PRINT
    END

```

```

// In the CPP file
m_wndToolBar.Create(this);
m_wndToolBar.LoadToolBar(IDR_TOOLBAR);

```

使用 TOOLBAR 资源时,只需改变资源语句中的数字,就能改变图像的大小。LoadToolBar 一个语句就能完成加载工具栏图像和设置按钮 ID 和按钮尺寸。要求 AppWizard 在应用程序中添加工具栏时,AppWizard 就是用这种方法定义工具栏的。

很幸运现在不必用手工编程方式创建和编辑 TOOLBAR 资源。AppWizard 向应用程序添加工具栏时,它创建一个 TOOLBAR 资源和与资源配套的位图。用 Visual C++ 的 Insert-Resource 命令也可以给项目添加 TOOLBAR 资源。一旦添加该资源后,就可以在 Visual C++ 的资源编辑器中编辑 TOOLBAR 资源和它的按钮位图了。

在默认方式下,工具栏按钮包含图像,不包含文本。用 CToolBar::SetButtonText 可在按钮表面添加字符串。给每个按钮指定文本后,用 CToolBar::SetSizes 可以调整按钮尺寸,使它适合字符串长度。下列语句基于 IDR_TOOLBAR 创建了一个工具栏,并在每个按钮表面添加了描述性文字:

```

// In the RC file
IDR_TOOLBAR BITMAP Toolbar.bmp

IDR_TOOLBAR TOOLBAR 40, 19
.
.
.

// In the CPP file
m_wndToolBar.Create(this);
m_wndToolBar.LoadToolBar(IDR_TOOLBAR);

m_wndToolBar.SetButtonText(0,_T("New"));
m_wndToolBar.SetButtonText(1,_T("Open"));
m_wndToolBar.SetButtonText(2,_T("Save"));
m_wndToolBar.SetButtonText(4,_T("Cut"));

```

```

m_wndToolBar.SetButtonText (5,_T ("Copy"));
m_wndToolBar.SetButtonText (6,_T ("Paste"));
m_wndToolBar.SetButtonText (7,_T ("Undo"));
m_wndToolBar.SetButtonText (9,_T ("Print"));

m_wndToolBar.SetSizes (CSize (48, 42), CSize (40, 19));

```

生成的工具栏请参见图 12-2。传递给 SetButtonText 的第一个参数指定了按钮的索引值,其中 0 代表工具栏左端位置上的按钮,1 代表右面的相邻按钮,如此类推。在添加按钮文本“后”而不是“前”,必须调用 SetSizes,否则按钮大小不能固定。而且按钮位图宽度也必须调整到能装下按钮文本。在该例中,Toolbar.bmp 中每个按钮位图的宽度设置为 40 个像素,高度设置为 19 个像素,使按钮形状近于正方形。

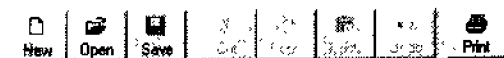


图 12-2 带有文字的工具栏按钮

除非您采取特殊措施,一般情况下工具栏按钮和标准按钮的动作相似:单击时,下陷;释放时,弹起。可以用 MFC 的 CToolBar::SetButtonStyle 函数创建复选按钮,使它们始终处于下陷状态直到再次单击之;或创建单选按钮,使它们一直处于下陷状态直到单击其他工具栏按钮。下列语句创建了一个正文格式化工具栏,它包括复选按钮,可用来选中黑体、斜体和带下划线正文;还包括单选按钮,可用来选中左对齐、居中或右对齐正文。

```

// In the RC file
IDR_TOOLBAR BITMAP Toolbar.bmp

IDR_TOOLBAR TOOLBAR 16, 15
BEGIN
    BUTTON ID_CHAR_BOLD
    BUTTON ID_CHAR_ITALIC
    BUTTON ID_CHAR_UNDERLINE
    SEPARATOR
    BUTTON ID_PARA_LEFT
    BUTTON ID_PARA_CENTER
    BUTTON ID_PARA_RIGHT
END

// In the CPP file
m_wndToolBar.Create (this);
m_wndToolBar.LoadToolBar (IDR_TOOLBAR);

m_wndToolBar.SetButtonStyle (0, TBBS_CHECKBOX);
m_wndToolBar.SetButtonStyle (1, TBBS_CHECKBOX);

```



```

m_wndToolBar.SetButtonStyle(7, TBBS_CHECKBOX);
m_wndToolBar.SetButtonStyle(4, TBBS_CHECKGROUP);
m_wndToolBar.SetButtonStyle(5, TBBS_CHECKGROUP);
m_wndToolBar.SetButtonStyle(6, TBBS_CHECKGROUP);

```

样式 TBBS_CHECKBOX 创建了一个复选按钮。TBBS_CHECKGROUP 相当于 TBBS_CHECKBOX|TBBS_GROUP, 创建了一个单选按钮。由于按钮 4、5 和 6 共享样式 TBBS_CHECKGROUP, 所以单击其中之一会“选中”该按钮, 并取消“选中”其他按钮。然而, 按钮 0、1 和 2 在操作中是彼此独立的, 只有在单击时, 它们才进行自身切换。通过 SetButtonStyle 还可以指定其他工具栏按钮样式, 有 TBBS_BUTTON(该样式可以创建标准按钮)和 TBBS_SEPARATOR(该样式可以创建按钮分隔标志)。补充的 CToolBar::GetButtonStyle 函数可以提取按钮样式。

给工具栏添加单选按钮时, 应在每组中选中一个成员作为默认按钮。下列代码在前一示例的基础上又选中 ID_PARA_LEFT 按钮作为默认按钮:

```

int nState =
    m_wndToolBar.GetToolBarCtrl().GetState(ID_PARA_LEFT);
m_wndToolBar.GetToolBarCtrl().SetState(ID_PARA_LEFT, nState |
    TBSTATE_CHECKED);

```

正如本章前部分介绍的那样, CToolBar::GetToolBarCtrl 返回对 CToolBarCtrl 的引用。类 CToolBarCtrl 为 CToolBar 提供了所有基本功能。CToolBarCtrl::GetState 返回工具栏按钮状态, CToolBarCtrl::SetState 则改变按钮状态。如果在传递给 SetState 的参数中设置 TBSTATE_CHECKED 标志, 则该按钮被选中。

实际上, 在 MFC 程序中, 如果处理程序改用 CCmdUI::SetCheck 实现选中或取消选中, 则通过这些新处理程序可以把标准按钮转换为复选按钮和单选按钮, 根本用不着 SetButtonStyle。以后会详细解释这一点。

12.1.2 固定式和浮动式工具栏

CToolBar 还有一个功能: 用户可以拖动工具栏, 使它脱离原位置, 并把它放在窗口的另一侧, 或让它浮动在自己的框架窗口中。您完全可以决定工具栏的摆放位置、摆放方式和浮动方式。您还可以创建自定义工具栏和静态工具调色板。自定义工具栏能按照用户的要求摆放、浮动和调整大小, 而静态工具调色板则始终浮动着, 并保持停放的行、列配置。

首次创建工具栏时, 它在框架窗口的一侧, 且不能移开。只有通过调用工具栏的 EnableDocking 函数 (CControlBar::EnableDocking) 和框架窗口的 EnableDocking 函数 (CFrameWnd::EnableDocking), 并由各自的位标志指定框架窗口一侧作为工具栏的安放位置, 工具栏的可浮动和可摆放性能才有效。表 12-1 中的值可以“或”起来传递给任一 EnableDocking 函数。

表 12-1 位标志值

位标志	说 明
CBRS_ALIGN_LEFT	允许停放在框架窗口左侧
CBRS_ALIGN_RIGHT	允许停放在框架窗口右侧
CBRS_ALIGN_TOP	允许停放在框架窗口顶部
CBRS_ALIGN_BOTTOM	允许停放在框架窗口底部
CBRS_ALIGN_ANY	允许停放在框架窗口任意一侧

在框架窗口类的成员函数中调用下列函数

```
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
```

可使 `m_wndToolBar` 代表的工具栏停放在父窗口的任意一侧。而

```
m_wndToolBar.EnableDocking(CBRS_ALIGN_TOP|CBRS_ALIGN_BOTTOM);
EnableDocking(CBRS_ALIGN_ANY);
```

则将工具栏放置在框架窗口的顶部或底部。工具栏和框架窗口都要指定停放位置,这似乎有些多余。但如果框架窗口包含多个工具栏,并且每个工具栏都有不同的停放的位置时,分别设定工具栏和框架窗口的摆放位置参数就显得很方便了。例如:如果 `m_wndToolBar1` 和 `m_wndToolBar2` 属于同一个框架窗口,则

```
m_wndToolBar1.EnableDocking(CBRS_ALIGN_TOP|CBRS_ALIGN_BOTTOM);
m_wndToolBar2.EnableDocking(CBRS_ALIGN_LEFT|CBRS_ALIGN_RIGHT);
EnableDocking(CBRS_ALIGN_ANY);
```

把 `m_wndToolBar1` 停放在顶部或底部,而把 `m_wndToolBar2` 停放在左侧或右侧。

用 `CFrameWnd` 的成员函数 `DockControlBar` 和 `FloatControlBar` 停放工具栏和使工具栏浮动。`DockControlBar` 将工具栏列放到它的父窗口。

```
DockControlBar(&m_wndToolBar);
```

将 `m_wndToolBar` 停放在它的默认位置——框架窗口顶部之内。而

```
DockControlBar(&m_wndToolBar, AFX_IDW_DOCKBAR_RIGHT);
```

将工具栏停放在框架窗口的右端。为了更好地摆放工具栏,可以给 `DockControlBar` 传递一个 `CRect` 对象或一个指向 `RECT` 结构的指针,其中 `RECT` 结构包含停放位置。在调用 `DockControlBar` 之前,即使已经用 `CControlBar::EnableDocking` 和 `CFrameWnd::EnableDocking` 使工具栏获得可摆放性,工具栏依旧不能脱离它的父窗口。

`FloatControlBar` 的功能正好和 `DockControlBar` 相反,调用它可使工具栏脱离自己的框架窗

口,并宣布工具栏可以浮动。如果用户拖动一个固定式工具栏,并把它停放在非停放位置,则主框架调用 `FloatControlBar` 使之成为浮动式工具栏。而通过 `CPoint` 设定工具栏左上角在屏幕坐标系中的位置,也可以使工具栏浮动起来:

```
FloatControlBar (&m_wndToolBar, CPoint (x, y));
```

还可以给 `FloatControlBar` 传递第3个参数。如果参数等于 `CBRS_ALIGN_TOP`,则工具栏沿水平方向放置;如果等于 `CBRS_ALIGN_LEFT`,则工具栏沿竖直方向放置。通过调用 `FloatControlBar` 创建的工具栏,它的初始状态是浮动的。如果以0调用 `EnableDocking`,而后再调用 `FloatControlBar`,就能得到一个浮动式工具栏,但无法将它停放到框架窗口一侧。MFC 编程人员有时利用这个技巧创建独立的工具调色板窗口。通过调用 `CControlBar::IsFloating`,可以在任意时刻判断工具栏是浮动的还是固定在某处的。通过调用工具栏的 `SetWindowText` 函数,还可以在浮动式工具栏的子框架窗口中添加标题。

在默认方式下,当浮动式工具栏被固定在框架窗口的顶部或底部时,工具栏沿水平方向排列;当被固定在框架窗口的左侧或右侧时,工具栏沿竖直方向排列;但在浮动过程中,它不再重新排列。通过在工具栏样式中添加 `CBRS_SIZE_DYNAMIC` 标志,可以使用户调整浮动式工具栏的大小。相反地,通过添加 `CBRS_SIZE_FIXED` 标志,可以保证(甚至在停放工具栏时)工具栏的尺寸和形状不变。`CBRS_SIZE_FIXED` 的另一个用处是创建具有固定行、列配置信息的浮动式工具调色板窗口。如果要创建包含多行按钮的静态工具调色板,则使用样式 `TBBS_WRAPPED` 告诉 `CToolBar` 换行符的位置。样式为 `TBBS_WRAPPED` 的工具栏按钮和文本文件中的回车符/换行符对相似,它后面的按钮放在新的一行。假定 `IDR_TOOLBAR` 代表一个包含9个按钮的工具栏,下列代码创建了一个固定的工具调色板窗口,包含3行,每行3个按钮:

```
m_wndToolBar.Create (this);
m_wndToolBar.LoadToolBar (IDR_TOOLBAR);
m_wndToolBar.SetBarStyle (m_wndToolBar.GetBarStyle () |
    CBRS_SIZE_FIXED);

m_wndToolBar.SetButtonStyle (2,
    m_wndToolBar.GetButtonStyle (0) | TBBS_WRAPPED);
m_wndToolBar.SetButtonStyle (5,
    m_wndToolBar.GetButtonStyle (0) | TBBS_WRAPPED);

EnableDocking (CBRS_ALIGN_ANY);
m_wndToolBar.EnableDocking (0);
FloatControlBar (&m_wndToolBar, CPoint (x, y));
```

给索引值为2和5的按钮添加 `TBBS_WRAPPED` 位,结果是每隔三个按钮创建一个换行符。而且因为工具调色板的 `EnableDocking` 函数是以0值调用的,所以工具调色板是浮动的,并且

不能固定到框架窗口。

如果应用程序有两个或更多个工具栏,则可以在工具栏的 `EnableDocking` 函数中添加 `CBRS_FLOAT_MULTI` 标志,并允许用户停放浮动式工具栏,最终形成混合型工具栏,使它们共享一个子框架窗口。不幸的是,样式 `CBRS_FLOAT_MULTI` 和 `CBRS_SIZE_DYNAMIC` 彼此不兼容,不能同时在一个工具栏中运用。

12.1.3 控制工具栏的可见性

大部分包含工具栏的应用程序都设有显示或隐藏工具栏的命令。MFC 应用程序可以用 `CFrameWnd` 的成员函数 `OnBarCheck` 来隐藏和显示工具栏。通过工具栏 ID 调用 `OnBarCheck`,如果工具栏处于显示状态,则该函数隐藏工具栏;如果工具栏被隐藏,则该函数显示工具栏。还有一个相关成员函数 `OnUpdateControlBarMenu`,通过选中或取消选中其 ID 和工具栏 ID 匹配的菜单项,它可以更新包含该切换命令的菜单。`OnBarCheck` 和 `OnUpdateControlBarMenu` 也适用于状态栏;您只需把工具栏 ID 更换为状态栏 ID 即可。

如果应用程序只有一个工具栏,并且具有默认 ID 值 `AFX_IDW_TOOLBAR`,则通过赋给菜单项特殊的 ID 值 `ID_VIEW_TOOLBAR`,所建菜单项可以切换工具栏。对于状态栏,则换成 `ID_VIEW_STATUS_BAR`。这里不再需要消息映射,因为 `CFrameWnd` 消息映射表中有消息映射项将这个“神奇”的菜单项 ID 映射为适当的 `CFrameWnd` 成员函数:

```
ON_UPDATE_COMMAND_UI (ID_VIEW_STATUS_BAR, OnUpdateControlBarMenu)
ON_COMMAND_EX (ID_VIEW_STATUS_BAR, OnBarCheck)
ON_UPDATE_COMMAND_UI (ID_VIEW_TOOLBAR, OnUpdateControlBarMenu)
ON_COMMAND_EX (ID_VIEW_TOOLBAR, OnBarCheck)
```

`ON_COMMAND_EX` 和 `ON_COMMAND` 相似, `ON_COMMAND_EX` 处理程序和 `ON_COMMAND` 处理程序的不同在于,前者接收 `UINT` 参数。该参数包含生成 `ON_COMMAND_EX` 消息的 UI 对象的 ID 值。`OnBarCheck` 认为工具栏 ID 和菜单项 ID 是一样的,并通过这个 ID 隐藏或显示工具栏。

如果您的应用程序使用一个工具栏,其 ID 不是 `AFX_IDW_TOOLBAR`,则有两种方法可以建立工具栏和命令间的联系,并更新决定工具栏可见性的处理程序。最简单的方法是赋给工具栏和相应菜单项相同的 ID,并在主框架窗口的消息映射表中将这个 ID 映射为 `OnBarCheck` 和 `OnUpdateControlBarMenu`。如果菜单项 ID 是 `ID_VIEW_TOOLBAR2`,消息映射项就是这样:

```
ON_UPDATE_COMMAND_UI (ID_VIEW_TOOLBAR2, OnUpdateControlBarMenu)
ON_COMMAND_EX (ID_VIEW_TOOLBAR2, OnBarCheck)
```

不要忘记,要使这种方法有效,工具栏的 ID “必须”和菜单项的 ID 相同。

第二种方法是编写自己的命令和更新处理程序,并用 `CFrameWnd::ShowControlBar` 隐藏

和显示工具栏。通过查看 `GetStyle` 返回值的 `WS_VISIBLE` 位,可以确定工具栏当前是否显示。

```
// In CMainFrame's message map
ON_COMMAND(ID_VIEW_TOOLBAR2, OnViewToolBar2)
ON_UPDATE_COMMAND_UI(ID_VIEW_TOOLBAR2, OnUpdateViewToolBar2UI)

.
.
.

void CMainFrame::OnViewToolBar2()
{
    ShowControlBar(&m_wndToolBar2, (m_wndToolBar2.GetStyle() &
        WS_VISIBLE) == 0, FALSE);
}

void CMainFrame::OnUpdateViewToolBar2UI(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck((m_wndToolBar2.GetStyle() &
        WS_VISIBLE) ? 1 : 0);
}
```

不要试图通过打开或关闭 `WS_VISIBLE` 标志切换工具栏,因为仅靠来回改换样式位是无法实现隐藏和显示工具栏的。例如:工具栏被显示或隐藏时(处于停放或浮动状态),MFC 会调整视图大小,以适应框架窗口客户区中可见区域内的变化。`ShowControlBar` 在隐藏或显示工具栏时会考虑到这些因素,以及其他有关因素。欲知详细情况,请参见 MFC 资源代码文件 `Winfrm.cpp` 中 `CFrameWnd::ShowControlBar` 的代码程序。

12.1.4 保持工具栏按钮和应用程序同步

在源代码中工具栏按钮和命令处理程序的连接方式与菜单项和命令处理程序的连接方式相同:都是通过消息映射表。同赋给菜单项更新处理程序一样,您可以赋给工具栏按钮更新处理程序。相同的 `CCmdUI` 函数既能更新菜单项,也能更新工具栏按钮。这正是 MFC 将指向 `CCmdUI` 的指针,而不是指向 `CMenu` 或 `CButton` 的指针,传递给更新处理程序的一个原因。在菜单更新期间调用 `CCmdUI::SetCheck` 可以选中或取消选中菜单项。在工具栏更新期间选中或取消选中按钮(工具栏按钮下陷或弹起)期间,调用的也是同一函数。因为 `CCmdUI` 抽炼出 UI 对象的实际特性,所以只要工具栏按钮和菜单项的 ID 相同,一个更新处理程序对两者都会起作用。

假定应用程序的 `Edit` 菜单中有 `Paste` 命令。剪贴板中有正文时,该命令有效;没有正文时,该命令无效。另外,假定应用程序还有 `Paste` 工具栏按钮,按钮和 `Edit-Paste` 的作用相同。菜单项和工具栏按钮都被赋予预定义命令 ID: `ID_EDIT_PASTE`,并且 `ID_EDIT_PASTE` 通过下列消息映射项映射到处理程序 `OnEditPaste`:

```
ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
```

如果每次显示 Edit 菜单时要更新 Paste 菜单项,则还需把 ID_EDIT_PASTE 映射为更新处理程序 OnUpdateEditPasteUI:

```
ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPasteUI)
```

OnUpdateEditPasteUI 在 `::IsClipboardFormatAvailable` 返回值的基础上借助于 `CCmdUI::Enable` 使 Paste 命令无效或有效:

```
void CMyClass::OnUpdateEditPasteUI (CCmdUI * pCmdUI)
{
    pCmdUI->Enable (::IsClipboardFormatAvailable (CF_TEXT));
}
```

在此基础之上,通过在 Edit 菜单中选中 Paste 或在工具栏中单击 Paste 按钮,就可以实现粘贴动作。另外,处理程序既然能保证菜单项和剪贴板状态同步,同样也能更新工具栏按钮。菜单项更新和工具栏更新的唯一区别是调用更新处理程序的时间。对菜单项来说,主框架在响应 `WM_INITMENUPOPUP` 消息时调用更新处理程序。对工具栏按钮来说,主框架在应用程序无消息可处理时调用更新处理程序。因此,尽管菜单更新要等到菜单快显示时才进行,但是工具栏按钮更新则是在状态发生变化后立刻进行的。这也有好处,因为工具栏按钮和菜单项不一样,它们始终显示在屏幕上。这个实际调用机制对应用程序来说是很明白的,只要给出更新处理程序,而后由主框架在需要的时候调用它们就行了。

前面曾提到:不必改变按钮样式,用更新处理程序就能创建复选按钮和单选按钮。这很容易实现:只要给出每个按钮的更新处理程序,并通过 `CCmdUI::SetCheck` 或 `CCmdUI::SetRadio` 实现选中或取消选中即可。如果按钮的命令处理程序来回在 `TRUE` 和 `FALSE` 间切换某个布尔变量,而且按钮的更新处理程序要参考该变量值决定选中或取消选中按钮,则按钮的动作和复选按钮相同。如果命令处理程序将该变量值设置成 `TRUE`,而将组中其他按钮的这个值设置成 `FALSE`,则按钮的动作和单选按钮相同。下列消息映射项、命令处理程序和更新处理程序创建了一组工具栏按钮。该组共三个按钮,其动作均和单选按钮相似。

```
// In CMyClass's message map
ON_COMMAND(ID_BUTTON1, OnButton1)
ON_COMMAND(ID_BUTTON2, OnButton2)
ON_COMMAND(ID_BUTTON3, OnButton3)
ON_UPDATE_COMMAND_UI(ID_BUTTON1, OnUpdateButton1)
ON_UPDATE_COMMAND_UI(ID_BUTTON2, OnUpdateButton2)
ON_UPDATE_COMMAND_UI(ID_BUTTON3, OnUpdateButton3)

*
*
*
```

```

void CMyClass::OnButton1 ()
{
    m_bButton1Down = TRUE;
    m_bButton2Down = FALSE;
    m_bButton3Down = FALSE;
}

void CMyClass::OnButton2 ()
{
    m_bButton1Down = FALSE;
    m_bButton2Down = TRUE;
    m_bButton3Down = FALSE;
}

void CMyClass::OnButton3 ()
{
    m_bButton1Down = FALSE;
    m_bButton2Down = FALSE;
    m_bButton3Down = TRUE;
}

void CMyClass::OnUpdateButton1 (CCommandUI * pCmdUI)
{
    pCmdUI->SetCheck (m_bButton1Down);
}

void CMyClass::OnUpdateButton2 (CCommandUI * pCmdUI)
{
    pCmdUI->SetCheck (m_bButton2Down);
}

void CMyClass::OnUpdateButton3 (CCommandUI * pCmdUI)
{
    pCmdUI->SetCheck (m_bButton3Down);
}

```

有了这些消息响应和更新处理函数,就不必考虑工具栏按钮是 TBBS_CHECKGROUP 按钮还是普通的 TBBS_BUTTON 按钮。单击其中之一,就可以把其他按钮状态变量的值设置成 FALSE,更新处理程序随后响应,画出新状态下的按钮。

12.1.5 添加工具提示和状态栏工具说明

工具栏最初出现在 Microsoft Windows 应用程序中时,它们有时不仅不能提供方便,反而成了一种障碍。这是因为按钮表面的图像不是总能清楚地表达按钮功能。一些 UI 设计者试图通过在按钮上添加文本来解决这个问题。还有一些人则更进一步,创建“工具提示”。

光标在工具栏按钮上停半秒左右时,屏幕上会出现工具提示小窗口,窗口中有描述性的文字,比如 Open 和 Paste 之类(参见图 12-3)。如今,工具提示在 Windows 应用程序中非常普遍。因为不必增大按钮尺寸,它们就能为可用的工具栏按钮提供与上下文有关的帮助,很好地解决了按钮功能不明确的问题。



图 12-3 显示有工具提示的浮动式工具栏

在 MFC 工具栏中添加工具提示很容易。只需在工具栏样式中添加 CBRSTOOLTIPS, 并创建包含工具提示文本的字符串表资源。字符串 ID 使工具提示和工具栏按钮相匹配。如果您使用标准的 MFC 命令 ID, 如 ID_FILE_OPEN 和 ID_EDIT_PASTE, 并在应用程序的 RC 文件中包含 Afxres.h, 则主框架会为您提供工具提示文本。对于其他命令 ID, 您则要通过赋给字符串资源与工具栏按钮 ID 相匹配的 ID 提供工具提示文本。下列代码创建了一个工具栏, 其中按钮可以实现常见的文本格式化, 同时还带有工具提示功能:

```
// In the RC file
IDR_TOOLBAR BITMAP Toolbar.bmp

IDR_TOOLBAR TOOLBAR 16, 15
BEGIN
    BUTTON ID_CHAR_BOLD
    BUTTON ID_CHAR_ITALIC
    BUTTON ID_CHAR_UNDERLINE
    SEPARATOR
    BUTTON ID_PARA_LEFT
    BUTTON ID_PARA_CENTER
    BUTTON ID_PARA_RIGHT
END

STRINGTABLE
BEGIN
    ID_CHAR_BOLD    "\nBold"
    ID_CHAR_ITALIC  "\nItalic"
    ID_CHAR_UNDERLINE "\nUnderline"
    ID_PARA_LEFT    "\nAlign Left"
    ID_PARA_CENTER  "\nAlign Center"
    ID_PARA_RIGHT   "\nAlign Right"
END

// In the CPP file
m_wndToolBar.Create(this, WS_CHILD|WS_VISIBLE |
```



```
CBRS_TOP|CBRS_TOOLTIPS);
m_wndToolBar.LoadToolBar(IDR_TOOLBAR);
```

光标停在工具栏按钮上时,如果有字符串资源的 ID 与该按钮的 ID 匹配,则在工具提示窗口中主框架把文本显示在换行符后面。光标移动时,工具提示消失。在过去,为了显示工具提示,还得设置计时器、监视鼠标移动以及子类窗口。如今,这些功能都提供给您了。

如果应用程序既有状态栏又有工具栏,通过在工具栏样式中设置 CBRS_FLYBY 位,则除(或不用)工具提示外,还可以使工具栏显示状态栏工具说明。掠过式文本是光标停在工具栏按钮上时显示在状态栏中的描述性文字。工具提示文本必须简短且中肯,但掠过式文本可以稍长些。您是不是想知道为什么前段字符串资源要用“\n”开头?这是因为同一个字符串资源确定了掠过式文本和工具提示文本。掠过式文本在换行符之前,而工具提示文本在换行符之后。如果上段代码还要包含掠过式文本,则修改后程序代码如下:

```
// In the RC file
IDR_TOOLBAR BITMAP Toolbar.bmp

IDR_TOOLBAR TOOLBAR 16, 15
BEGIN
    BUTTON ID_CHAR_BOLD
    BUTTON ID_CHAR_ITALIC
    BUTTON ID_CHAR_UNDERLINE
    SEPARATOR
    BUTTON ID_PARA_LEFT
    BUTTON ID_PARA_CENTER
    BUTTON ID_PARA_RIGHT
END

STRINGTABLE
BEGIN
    ID_CHAR_BOLD "Toggle boldface on or off\nBold"
    ID_CHAR_ITALIC "Toggle italic on or off\nItalic"
    ID_CHAR_UNDERLINE "Toggle underline on or off\nUnderline"
    ID_PARA_LEFT "Align text flush left\nAlign Left"
    ID_PARA_CENTER "Center text between margins\nAlign Center"
    ID_PARA_RIGHT "Align text flush right\nAlign Right"
END

// In the CPP file
m_wndToolBar.Create(this, WS_CHILD|WS_VISIBLE);
    CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY);
m_wndToolBar.LoadToolBar(IDR_TOOLBAR);
```

如果菜单项和工具栏按钮的 ID 相同,则在菜单项加亮显示的同时,相应字符串资源中的文本也会显示,并且该文本出现在换行符之前。很快我们会介绍到状态栏的这个特性,

当然还有其他特性。

借助于 Visual C++ 的资源编辑器,您可以通过可视化操作将工具提示和掠过式文本赋给工具栏按钮。先打开工具栏资源,然后双击工具栏按钮,这时会显示该按钮的属性表。然后在 Prompt 框中敲入一个字符串,并把得到的掠过式文本、工具提示文本或两个文本赋给按钮,如图 12-4 所示。

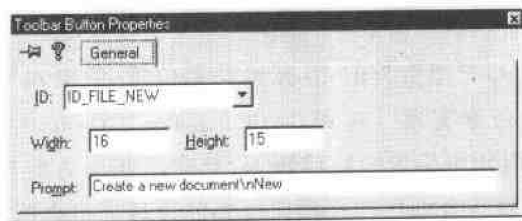


图 12-4 赋给工具栏按钮工具提示和掠过式文本

12.1.6 在工具栏中添加非按钮控件

在工具栏中按钮要比其他控件类型常见得多,但是 CToolBars 也接受非按钮控件,如组合框和复选框。假定您想在工具栏中添加组合框,使用户可以在下拉列表中选择打字机字体、字体大小或其他设置。您可以这样来实现。

第 1 步,先在 TOOLBAR 资源中添加一个按钮分隔符或空白按钮。该按钮可具有任意命令 ID 和按钮图像。下面的 TOOLBAR 资源定义中,分隔符作为组合框的预留位置,使组合框将来出现在最后一个按钮的右侧:

```
IDR_TOOLBAR TOOLBAR 16, 15
BEGIN
    BUTTON ID_CHAR_BOLD
    BUTTON ID_CHAR_ITALIC
    BUTTON ID_CHAR_UNDERLINE
    SEPARATOR
    BUTTON ID_PARA_LEFT
    BUTTON ID_PARA_CENTER
    BUTTON ID_PARA_RIGHT
    SEPARATOR // Space between button and combo box
    SEPARATOR // Placeholder for combo box
END
```

第 2 步,调用 CToolBar::SetButtonInfo 来增加预留位置的宽度,使它能容纳组合框,最后在该空白处创建一个组合框。假定工具栏是由通过 CToolBar 派生的类表示的,并且 m_wnd-ComboBox 是该派生类的 CComboBox 数据成员, IDC_COMBOBOX 为组合框的控件 ID, nWidth 和 nHeight 为指定的组合框尺寸,则下面代码节选自派生类的 OnCreate 处理程序,它能帮助

您了解如何创建组合框：

```
SetButtonInfo(8, IDC_COMBOBOX, TBBS_SEPARATOR, nWidth);
CRect rect;
GetItemRect(8, &rect);
rect.bottom = rect.top + nHeight;
m_wndComboBox.Create(WS_CHILD|WS_VISIBLE|WS_VSCROLL
    CBS_SORT|CBS_DROPDOWNLIST, rect, this, IDC_COMBOBOX);
```

调用 `CToolBar::SetButtonInfo` 把组合框的 ID 赋给预留位置,并沿水平方向拓展预留位置,使它的宽度等于指定的组合框宽度。在调用 `CComboBox::Create` 创建组合框之前,先调用 `CToolBar::GetItemRect` 获取预留位置的控件矩形,然后沿竖直方向拉伸该矩形,直到能容纳组合框的列表框部分,这样组合框出现在预留位置的顶端。组合框隶属于工具栏,所以工具栏移动时,组合框也随之移动。工具栏还接收组合框的 `WM_COMMAND` 消息,但是因为命令分配机制,框架窗口、视图和其他标准命令目标也可以处理组合框发送给它父窗口的通知。

非按钮控件的工具提示和掠过式文本又是怎样的呢?对主框架来说,组合框只是工具栏中的另一个控件,并且也可以和按钮一样拥有工具提示和掠过式文本。如果要给组合框添加工具提示和掠过式文本,您所做的是定义一个字符串资源,其 ID 为 `IDC_COMBOBOX`。当光标停在组合框上时,工具提示窗口会自动出现,而掠过式文本也会出现在状态栏中。

12.1.7 更新非按钮控件

由于 `CCmdUI` 不是为处理组合框而设计的,所以把更新处理程序赋给工具栏中的组合框没有任何意义。但是 MFC 提供了另一种适合非按钮控件的更新机制。`CControlBar::OnUpdateCmdUI` 是一个虚函数,主结构在 CPU 空闲时调用它。派生的工具栏类可以重载 `OnUpdateCmdUI`,并在此更新那些没有 UI 更新处理程序的控件。`OnUpdateCmdUI` 是保证自定义工具栏控件和应用程序其他部分同步的最佳方法,而且这种被动实现方式和用于工具栏和菜单项的更新机制非常相近。

如果您已经从 `CToolBar` 派生出一个工具栏类 `CStyleBar`,其中包含具有系统字体列表的组合框。如果您希望用户在文档中移动插入符时,组合框能随时更新,使选中项为当前插入符所在位置的字体,则不必通过直接更新组合框的选中项来响应插入符位置的每一个变化,可以按照下面方式重载 `OnUpdateCmdUI`:

```
void CStyleBar::OnUpdateCmdUI(CFrameWnd* pTarget,
    BOOL bDisableIfNoHandler)
{
    .
    .
    .
}
```

```

CToolBar::OnUpdateCmdUI (pTarget, bDisableIfNoHandler);
CString string = GetTypefaceAtCaret ();
if (m_wndComboBox.SelectString ( 1, string) == CB_ERR)
    m_wndComboBox.SetCurSel (-1);

```

GetTypefaceAtCaret 是 CStyleBar 的辅助函数,它提取文档或视图中的字体信息,并返回含有字体名的字符串。GetTypefaceAtCaret 返回后,调用 CComboBox::SelectString 选中相应的组合框项。如果 SelectString 调用失败,则以参数 -1 调用 CComboBox::SetCurSel 来清空组合框。有了这个简单的更新处理程序,当用户在文档中移动光标时,组合框选项就会和插入符保持同步。本章后面给出的 MyWord 应用程序使用了相近的 OnUpdateCmdUI 处理程序,实现了一对组合框和插入符位置的同步。其中一个组合框用于字体,另一个用于字体大小。

一般可忽略传递给 OnUpdateCmdUI 的参数 pTarget 和 bDisableIfNoHandler。但是一定要在派生类的 OnUpdateCmdUI 函数中调用 CToolBar::OnUpdateCmdUI,避免和传统工具栏按钮的更新处理程序冲突。

12.1.8 使工具栏设置永久化

MFC 提供了两个有用的函数: CFrameWnd::SaveBarState 和 CFrameWnd::LoadBarState,它们可以保存会话过程中形成的工具栏设置。SaveBarState 将每个工具栏的停放或浮动状态、位置、方向和可见性情况写到注册表或私有 INI 文件中(在 Windows 95 和 Windows 98 以及所有版本的 Windows NT 中,如果 SaveBarState 要使用注册表,则应该在应用程序类的 InitInstance 函数中调用 CWinApp::SetRegistryKey)。如果应用程序包含状态栏,SaveBarState 也会记录有关状态栏的信息。应用程序重新启动时,调用 LoadBarState 将设置信息读出,并将每个工具栏和状态栏恢复到以前的状态。一般创建工具栏和状态栏后,在主框架窗口的 OnCreate 函数中调用 LoadBarState,而 SaveBarState 是在框架窗口的 OnClose 函数中被调用的。在应用程序运行时如果 Windows 被关闭,这时如果您想保存控件栏的设置,则还要在 OnEndSession 函数中调用 SaveBarState。

如果要同时保存浮动式工具栏和停放式工具栏的状态,则不能在框架窗口的 OnDestroy 函数中调用 SaveBarState。停放式工具栏是所在框架窗口的子窗口,而浮动式工具栏是围绕它的小型框架窗口的子窗口。小型框架窗口是框架窗口的弹出式窗口,但不是框架窗口的子窗口(弹出式窗口是样式为 WS_POPUP 的窗口;而子窗口是样式为 WS_CHILD 的窗口)。这个区别非常重要,因为弹出式窗口是在框架窗口之前被清除的,而子窗口则是在它们的父窗口之“后”被清除的。如果调用了框架窗口的 OnDestroy 函数,浮动式工具栏就不再存在了。因此,如果在 OnDestroy 中调用 SaveBarState,对于没有停放在框架窗口中的工具栏,SaveBarState 就无法保存状态信息。

12.1.9 AppWizard 提供的工具栏支持

可以用 AppWizard 在应用程序中添加基本的工具栏。在 AppWizard 的 Step 4 对话框中选中 Docking Toolbar 框(如图 12-5 所示),添加一个简单的工具栏,其中包含命令按钮 File-Open、File-Save 和其他常用命令。除了创建 TOOLBAR 资源和按钮位图,AppWizard 还在主框架窗口类中添加 CToolBar 对象 m_wndToolBar,并在框架窗口的 OnCreate 函数中包含创建工具栏和使停放有效的代码。

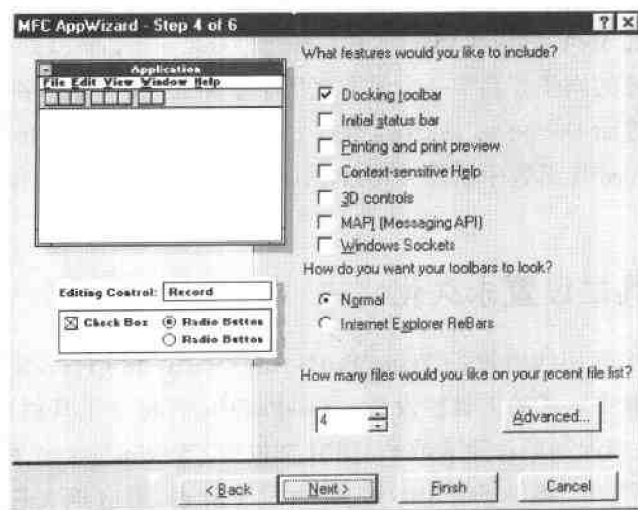


图 12-5 使用 AppWizard 添加工具栏

AppWizard 在工具栏创建程序中使用了 `CToolBar::CreateEx`, 而不是 `CToolBar::Create`, 同时它将 `CBRS_GRIPPER` 和 `TBSTYLE_FLAT` 标志传递给 `CreateEx`。标志 `CBRS_GRIPPER` 用于在工具栏的左边缘画一个细竖杠或“夹子”。标志 `TBSTYLE_FLAT` 用于创建一个“扁平”工具栏,其按钮的边只有在光标停在按钮上时才会显示出来,这和 Visual C++ 中的那些按钮一样。只有安装了 Internet Explorer 的系统才支持扁平工具栏。而在较早的系统中则采用普通工具栏的外观。

12.2 状态栏

状态栏可以显示与上下文有关的工具栏按钮和菜单项的帮助信息。状态栏在 Windows 应用程序中已变得非常普遍。SDK 的 Windows 应用程序通常通过捕获 `WM_MENUSELECT` 消息来更新状态栏的描述性帮助文本。MFC 提供的方法则更加容易。当 `CStatusBar` 被连接

到框架窗口时,如果有菜单项被加亮显示,CStatusBar 则自动显示帮助文本。如果应用程序包含一个工具栏,而该工具栏的样式又包含 CBR_S_FLYBY 标志,则状态栏也显示工具栏按钮的帮助文本。除了要创建和初始化状态栏(只需几行代码即可),用户需要做的仅是:在应用程序的资源文件中以字符串资源形式提供帮助文本。主框架则完成剩下的工作。

当然,状态栏不止能显示帮助文本。状态栏可以划分为一个或多个区域,它们分别对应着窗格、面板或指示符。可以单独设置每个窗格的文本,这样,一个窗格可以显示文档的当前行、页号,另一个则可以显示菜单和工具栏的帮助,其他的还可以显示当前 Caps Lock 和 Num Lock 的状态。一些状态栏甚至包含进度控件,以显示如保存、加载文档等较长的操作中所完成任务的比例。

12.2.1 创建和初始化状态栏

在 MFC 中,状态栏是 CStatusBar 的一个实例。使用状态栏的应用程序一定要将 CStatusBar 对象声明为框架窗口类的成员。而后,在框架窗口的 OnCreate 函数中用一条语句创建状态栏:

```
m_wndStatusBar.Create(this);
```

传递给 Create 的唯一参数确定了状态栏的父窗口。如果传递的参数是指向框架窗口的 this 指针,则状态栏成为框架窗口的子窗口。用这种方式创建的状态栏不必在应用程序结束前手工清除,因为清除它的父窗口时,它会自动清除。CStatusBar::Create 还接受确定状态栏样式和子窗口 ID 的参数,但 MFC 提供了这些参数的默认值,并且这些默认值对大部分应用程序来说都很合适。

状态栏创建后,调用 CStatusBar::SetIndicators 将状态栏初始化。SetIndicators 确定状态栏包含的窗格数,并同时赋给各窗格字符串资源。语句

```
HINT nIndicator = ID_SEPARATOR;
m_wndStatusBar.Create(this);
m_wndStatusBar.SetIndicators(&nIndicator, 1);
```

创建了一个只有一个窗格的状态栏。ID_SEPARATOR 是一个通用 ID,说明没有字符串资源和这个窗格关联。您可以创建一个简单的“二进制”窗格,用于指示应用程序的某个功能是否已打开,实现步骤为:指定字符串资源 ID、连接窗格和更新函数。其中更新函数使用了 CCmdUI::Enable 使窗格有效或失效。有效的窗格显示赋予它的字符串资源,失效的窗格则是空白。下列代码创建的状态栏包含一个窗格,当应用程序处于插入模式时,窗格显示文本字符“INS”,而在加粗模式时,则什么都不显示。这个示例假定 m_bInsert 为非零值时,插入模式打开,m_bInsert 为零时,插入模式关闭:

```
// In the RC file
```

```

STRINGTABLE
BEGIN
    ID_INDICATOR_INS "INS"
END

// In CMainFrame's message map
ON_UPDATE_COMMAND_UI (ID_INDICATOR_INS, OnUpdateIndicator)

// In CMainFrame::OnCreate
static UINT nIndicators[] = {
    ID_SEPARATOR,
    ID_INDICATOR_INS
};

m_wndStatusBar.Create (this);
m_wndStatusBar.SetIndicators (nIndicators, 2);

// Elsewhere in CMainFrame
void CMainFrame::OnUpdateIndicator (CCmdUI * pCmdUI)
{
    pCmdUI->Enable (m_bInsert);
}

```

在这个示例中,框架窗口处理 UI 更新命令。在实际应用程序中,把 `OnUpdateIndicator` 作为文档或视图类的成员函数更合适。`ID_INDICATOR_INS` 是应用程序在其他部分定义的符号常量,MFC 不会自动实现。

如果状态栏窗格要显示键盘状态,MFC 为它们定义了 4 种特殊的指示符 ID,映射到 `CFrameWnd` 类的 4 个公用更新函数:

- `ID_INDICATOR_CAPS`,对应于 Caps Lock 键
- `ID_INDICATOR_NUM`,对应于 Num Lock 键
- `ID_INDICATOR_SCRL`,对应于 Scroll Lock 键
- `ID_INDICATOR_KANA`,对应于日本式键盘的 Kana 键

当 Caps Lock 打开时,ID 值为 `ID_INDICATOR_CAPS` 的状态栏窗格显示 CAP。类似地,Num Lock 打开时,`ID_INDICATOR_NUM` 窗格显示 NUM;Scroll Lock 打开时,`ID_INDICATOR_SCRL` 窗格显示 SCRL;而当日本式键盘上的 Kana 模式打开时,`ID_INDICATOR_KANA` 窗格显示 ANA。主框架(实际上是 `CFrameWnd::OnUpdateKeyIndicator`) 保持这些指示符和键盘同步。因此,如果要创建一个具有 Caps Lock、Num Lock 和 Scroll Lock 指示符的状态栏,只要在传递给 `SetIndicators` 的数组中添加相应的 ID 值即可。

```

static UINT nIndicators[] = {
    ID_SEPARATOR,
    ID_INDICATOR_CAPS,

```

```

    ID_INDICATOR_NUM,
    ID_INDICATOR_SCROLL,
};

m_wndStatusBar.Create(this);
m_wndStatusBar.SetIndicators(nIndicators, 4);

```

生成的状态栏请参见图 12-6。空白窗格表示 Scroll Lock 未打开。CStatusBar 自动将窗格从状态栏的最右端开始排列,并把最左面的窗格延伸,直到充满剩余的状态栏空间。另外,它还调整其他窗格的大小,使它们能正好显示各自的字符串。除第一个窗格外,其他窗格都是“嵌入式”的,这使得它们即使为空白时也可见。

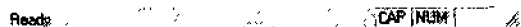


图 12-6 具有 Caps Lock、Num Lock 和 Scroll Lock 指示符的状态栏

12.2.2 为菜单项提供上下文相关帮助

如果您给第一个窗格(左端的)赋予 ID 值 ID_SEPARATOR,这就等于启用了 MFC 的一个特殊功能,这个功能在设计上非常优美而简洁。当用户加亮显示某菜单项时,主框架会查看并确定应用程序的 EXE 文件是否包含 ID 值等于菜单项 ID 的字符串资源。如果找到这样的匹配,则加载字符串资源并把它显示在状态栏窗格中。因此,通过提供 ID 为菜单项 ID 的字符串资源,您可以给应用程序菜单提供与上下文有关的帮助。如果菜单项和工具栏按钮的 ID 相同,则字符串资源既可以作为菜单项的帮助文本,也可以作为工具栏的掠过式文本。

和工具栏按钮一样,主框架还为 ID_FILE_NEW、ID_FILE_OPEN 和其他常用命令 ID 提供默认帮助字符串。同时主框架还为系统菜单中的某些命令提供默认帮助字符串(要查看预定义的 ID 和相关的帮助文本和工具提示文本,可参见 MFC 资源代码文件 Prompts.rc)。只要在应用程序资源文件中包含头文件 Afxres.h,主框架的预定义字符串资源便也在其中了。如果用 AppWizard 创建应用程序,则自动包含了 Afxres.h。不必手工编程添加其他菜单项的字符串资源,在菜单编辑器中双击某菜单项,并在 Menu Item Properties 窗口的 Prompt 框中输入字符串即可。

通过定义相同 ID 的字符串资源,可以覆盖预定义菜单项 ID 的帮助文本。如果要想做得好,可在应用程序的字符串表中添加语句

```
AFX_IDS_IDLEMESSAGE "Ready"
```

当没有菜单拉下或菜单项选中时,主框架会在状态栏中显示“Ready”。如果使用 AppWizard 在应用程序中添加状态栏,这已是自动完成的。

12.2.3 创建自定义状态栏窗格

现在您已经知道如何综合使用字符串资源和更新函数在状态栏中显示帮助文本,添加

Caps Lock、Num Lock 和 Scroll Lock 指示符,以及创建简单的打开/关闭指示符了。但对于比较复杂的状态栏(比如:在 Microsoft Word、Microsoft Excel、Microsoft PowerPoint 和其他 Windows 应用程序中见到的那些)又该怎么办呢?例如:怎样创建一个状态栏窗格,让它显示日期和当前页码呢?

对于初学者,可以在状态栏中添加窗格,并利用 CStatusBar 的 SetPaneInfo 函数随意调整窗格大小。SetPaneInfo 的 4 个参数按顺序为:窗格的索引、窗格的 ID、样式和宽度。窗格样式是表 12-2 中值的组合。

表 12-2 样式值

样式	说 明
SBPS_NOBORDERS	不具备 3-D 边界
SBPS_POPOUT	具备凸起的边界
SBPS_NORMAL	不扩展面板宽度,具备凸起或内陷的边界
SBPS_DISABLED	窗格中不显示文本
SBPS_STRETCH	将状态栏剩余空间并入该窗格,每个状态栏只能有一个窗格具备此样式
SBPS_OWNERDRAW	创建者的窗格

下列代码创建了一个状态栏,它具有三个自定义窗格。第一个窗格的宽度为 64 像素,不具备 3-D 边界。第二个的宽度也是 64 像素,具有凸起边界。第三个的宽度可变,为状态栏的剩余空间。

```
static UINT nIndicators[] = {
    ID_SEPARATOR,
    ID_SEPARATOR,
    ID_SEPARATOR
};

m_wndStatusBar.Create(this);
m_wndStatusBar.SetIndicators(nIndicators, 3);

m_wndStatusBar.SetPaneInfo(0, ID_SEPARATOR, SBPS_NOBORDERS, 64);
m_wndStatusBar.SetPaneInfo(1, ID_SEPARATOR, SBPS_POPOUT, 64);
m_wndStatusBar.SetPaneInfo(2, ID_SEPARATOR, SBPS_NORMAL |
    SBPS_STRETCH, 0);
```

在实际应用程序中,您可能不想费力地确定像素,而想用另一种可衡量屏幕大小的单位,如状态栏字体中字符的平均宽度,来表示窗格的宽度。调用 CStatusBar 从 CWnd 继承来的 GetFont 函数,可得到默认状态栏字体的 CFont 指针。

自定义窗格创建后,需要告知状态栏窗格要显示什么。有两种方法可以在窗格中显示文本:一是调用 CStatusBar::SetPaneText 直接设置文本,二是给窗格映射一个更新函数,并由

此函数调用 `CCmdUI::SetText` 设定文本。到底使用哪种方法取决于您需要怎样更新窗格。下面的程序段设置了一个每 200 毫秒发一次消息的计数器,并用 `SetPaneText` 更新窗格 2 显示的时、分、秒(Windows 计时器将在第 14 章介绍)。此时,`SetPaneText` 用索引值指定窗格。

```
// In CMainFrame::OnCreate
SetTimer (ID_TIMER, 200, NULL);

.
.
.

void CMainFrame::OnTimer (UINT nTimerID)
{
    CTime time = CTime::GetCurrentTime();
    int nSecond = time.GetSecond();
    int nMinute = time.GetMinute();
    int nHour = time.GetHour() % 12;

    CString string;
    string.Format (_T ("%0.2d:%0.2d:%0.2d"), nHour, nMinute, nSecond);
    m_wndStatusBar.SetPaneText (2, string);
}
;
```

另一种方法是:赋给窗格特有的 ID,如 `ID_INDICATOR_TIME`,并通过消息映射将它和更新函数联系起来。现在,主框架就会不断更新显示在状态栏中的时间了。

```
// In the message map
ON_UPDATE_COMMAND_UI (ID_INDICATOR_TIME, OnUpdateTime)

.
.
.

void CMainFrame::OnUpdateTime (CCmdUI * pCmdUI)
{
    CTime time = CTime::GetCurrentTime();
    int nSecond = time.GetSecond();
    int nMinute = time.GetMinute();
    int nHour = time.GetHour() % 12;

    CString string;
    string.Format (_T ("%0.2d:%0.2d:%0.2d"), nHour, nMinute, nSecond);
    pCmdUI->SetText (string);
}
;
```

定义 `ID_INDICATOR_TIME` 的最好方法是:在应用程序中添加一个具有该 ID 的字符串资源。如果赋给字符串一个虚设值,如“MMMMM”,则 MFC 用该字符串的宽度调整状态栏窗格的大小。顺便提一句,如果在写往状态栏的文本前加一个制表符(“\t”),则窗格文本会居

中显示;如果加两个制表符(“\t\t”),则窗格文本会右对齐显示。

12.2.4 AppWizard 提供的状态栏支持

您可以使用 AppWizard 在 MFC 应用程序中添加状态栏。首先,在 AppWizard 的 Step 4 对话框中选中 Initial Status Bar 框,如图 12-7 所示。AppWizard 自动在主框架窗口类中添加一个 CStatusBar 成员变量,并在 OnCreate 函数中创建状态栏。它创建的状态栏有 4 个窗格:ID_SEPARATOR 窗格(用于显示帮助文本),以及相应于 Caps Lock、Num Lock 和 Scroll Lock 键的指示符窗格。

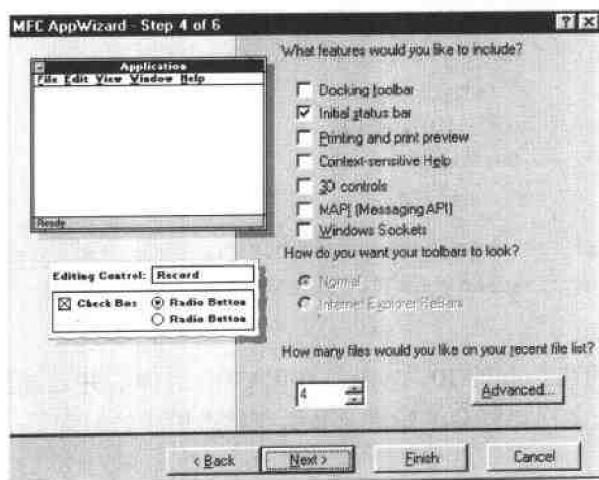


图 12-7 使用 AppWizard 添加状态栏

关于 AppWizard 自动生成的状态栏,初级 MFC 编程人员可能会问的一个问题是:怎样才能取消键盘指示符窗格?很简单,首先在 AppWizard 生成的主框架窗口类的 CPP 文件中找到下列语句:

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

然后去掉最后 3 项,数组变成:

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

```
ID_SEPARATOR    // status line indicator
};
```

重新编译应用程序,指示符窗格就没有了。

12.3 总结: MYWORD 应用程序

图 12-8 中的示例程序用到了前面介绍的许多原则。MyWord 是基于 CRichEditView 编制的小型文字处理程序。MFC 的 CRichEditView 类比 CEditView 的功能更强大,它有公用控件库提供的丰富的文本编辑控件的支持。它具有高级文本格式化功能,并只需一个简单的函数调用就能读、写复文本格式(RTF)文件。MyWord 没有用到 CRichEditView 的所有功能,实际上,它仅仅涉及到一些基本功能(如果要详细了解 CRichEditView,请参见 MFC 提供的 Wordpad 示例程序。Wordpad 文件是 Windows 中 Wordpad 小应用程序的真实源代码)。但是 MyWord 又比只有几百行的程序深入些,对于编写自己的基于 CRichEditView 的应用程序,它是很好的起点。

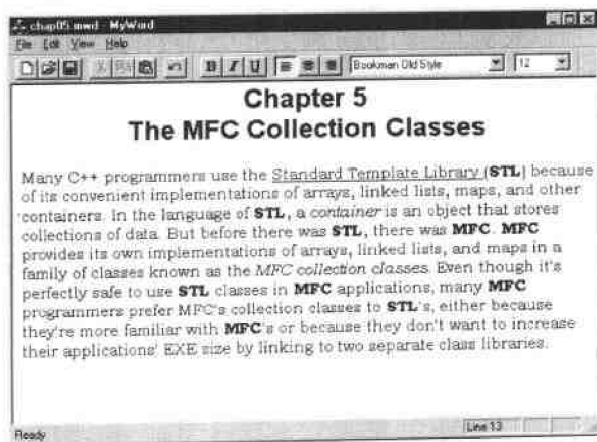


图 12-8 MyWord 窗口

MyWord 使用了两个工具栏和一个状态栏。主要的工具栏包括以下命令的快捷方式按钮: File 菜单的 New、Open 和 Save 菜单项; Edit 菜单的 Cut、Copy、Paste 和 Undo 菜单项。另一个工具栏,我称它为“样式栏”,包含设置字符格式(黑体、斜体和下划线体)的复选按钮,设置段落对齐的单选按钮(左对齐、居中和右对齐)和选择字体和字体大小的组合框。两个工具栏都可以脱离框架窗口,成为浮动的,并停放在其他位置;而且在浮动时两个工具栏都能调整大小。试试看: 将主工具栏拖到窗口右侧,并沿竖直方向放置它。移动样式栏并在窗口中间释放它,使它成为浮动式调色板。使用 View 菜单隐藏和显示工具栏和状态栏。还可以通过单击小型框架窗口中的关闭按钮隐藏工具栏。该工具栏脱离主框架窗口后,便浮动在

小型框架窗口中。如果要重新显示该工具栏,只需在 View 菜单中选中 Toolbar 或 Style Bar 命令。

位于 MyWord 框架窗口底部的状态栏显示菜单项和工具栏控件的帮助文本。它包含 Caps Lock 和 Num Lock 指示符以及行号。行号随文档中插入符的移动不断更新。Caps Lock 和 Num Lock 指示符是使用 MFC 的预定义 ID ID_INDICATOR_CAPS 和 ID_INDICATOR_NUM 添加的。行号指示符是通过 ON_UPDATE_COMMAND_UI 处理程序更新的。调用该处理程序可以在 CRichEditView 中提取当前行号,用公式表示含有该行号的字符串,并用 CCmdUI::SetText 更新状态栏显示。调整行号窗格大小,直至适合虚设字符串“Line 00000”,其资源 ID (ID_INDICATOR_LINE)和状态栏的 ID 一样。您不会看到虚设字符串,因为在状态栏显示之前窗格就已经更新为真实的行号了。

现在使用 AppWizard 编制 MyWord。在 AppWizard 的 Step 4 对话框中,选中 Docking Toolbar 和 Initial Status Bar,并在 Step 6 对话框中选中 CRichEditView 作为视图的基类。然后,派生得到代表样式栏的 CStyleBar 类,在框架窗口类中添加一个 CStyleBar 数据成员,并修改框架窗口的 OnCreate 函数,创建样式栏。(我使用 ClassWizard 实现类的派生,由于 ClassWizard 不支持 CToolBar 作基类,所以我从 CCmdTarget 派生出 CStyleBar,然后通过手工修改将基类改成 CToolBar。)我用 Visual C++ 的 Insert-Resource 命令创建工具栏资源。样式栏就是在这个工具栏资源基础上创立的。然后我在工具栏编辑器中添加按钮。而完成 MyWord 则要编写命令处理程序、更新处理程序和普通的类成员函数,这才是应用程序的核心。

MyWord 中框架窗口、文档、视图和样式栏类的资源代码列在图 12-9 中。花些时间看看工具栏和状态栏是如何处理的。然后向前翻几页再仔细阅读资源代码的关键部分。

MainFrm.h

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

#ifndef __AFX_MAINFRM_H__C85C9089_A154_11D2_8E53_006008A82731__INCLUDED_
#define __AFX_MAINFRM_H__C85C9089_A154_11D2_8E53_006008A82731__INCLUDED

#include "StyleBar.h" // Added by ClassView
#ifdef _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
    CStyleBar m_wndStyleBar;
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;
// Generated message map functions
protected:
    BOOL CreateToolBar();
    BOOL CreateStyleBar();
    BOOL CreateStatusBar();
    //||AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    //||AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

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

#endif
// !defined(AFX_MAINFRM_H__C85C9089_A154_11D2_8B53_006008A82731__INCLUDED_)

```

MainFrm.cpp

```

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

#include "stdafx.h"
#include "MyWord.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_CLOSE()
    //}}AFX_MSG_MAP

    ON_COMMAND_EX (IDW_STYLE_BAR, OnBarCheck)
    ON_UPDATE_COMMAND_UI (IDW_STYLE_BAR, OnUpdateControlBarMenu)
END_MESSAGE_MAP()

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

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

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

    //
    // Tell the frame window to permit docking.
    //

```

```

        EnableDocking(CBRS_ALIGN_ANY);

        //
        // Create the toolbar, style bar, and status bar.
        //
        if (!CreateToolBar())
            !CreateStyleBar()
            !CreateStatusBar()
            return -1;

        //
        // Load the saved bar state (if any).
        //
        LoadBarState(_T("MainBarState"));
        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

    void CMainFrame::OnClose()
    {
        SaveBarState(_T("MainBarState"));
        CFrameWnd::OnClose();
    }

    BOOL CMainFrame::CreateToolBar()

```



```

    {
        if (!m_wndToolBar.Create(this) ||
            !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
            return FALSE;

        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
            CBRS_TOOLTIP|CBRS_FLYBY|CBRS_SIZE_DYNAMIC);

        m_wndToolBar.SetWindowText(_T("Main"));
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);
        return TRUE;
    }

    BOOL CMainFrame::CreateStyleBar()
    {
        if (!m_wndStyleBar.Create(this, WS_CHILD|WS_VISIBLE|CBRS_TOP |
            CBRS_TOOLTIPS|CBRS_FLYBY|CBRS_SIZE_DYNAMIC, IDW_STYLE_BAR))
            return FALSE;

        m_wndStyleBar.SetWindowText(_T("Styles"));
        m_wndStyleBar.EnableDocking(CBRS_ALIGN_TOP|CBRS_ALIGN_BOTTOM);
        DockControlBar(&m_wndStyleBar);
        return TRUE;
    }

    BOOL CMainFrame::CreateStatusBar()
    {
        static UINT nIndicators[] = {
            ID_SEPARATOR,
            ID_INDICATOR_LINE,
            ID_INDICATOR_CAPS,
            ID_INDICATOR_NUM
        };

        if (!m_wndStatusBar.Create(this))
            return FALSE;

        m_wndStatusBar.SetIndicators(nIndicators, 4);
        return TRUE;
    }

```

MyWordDoc.h

```

// MyWordDoc.h : interface of the CMyWordDoc class
//
////////////////////////////////////

```

```

#ifndef defined(
    AFX_MYWORDDOC_H__C85C908B_A154_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_MYWORDDOC_H__C85C908B_A154_11D2_8E53_006008A82731__INCLUDED_

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

class CMyWordDoc : public CRichEditDoc
{
protected: // create from serialization only
    CMyWordDoc();
    DECLARE_DYNCREATE(CMyWordDoc)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMyWordDoc)
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //{{AFX_VIRTUAL
    virtual CRichEditCtrlItem* CreateClientItem(RXOBJECT* preo) const;

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

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CMyWordDoc)
        // 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_MYWORDDOC_H_ C85C908E_A154_11D2_8E53_006008A82731_ .INCLUDED. )

```

MyWordDoc.cpp

```

// MyWordDoc.cpp : implementation of the CMyWordDoc class
//

#include "stdafx.h"
#include "MyWord.h"

#include "MyWordDoc.h"
#include "CntrIter.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CMyWordDoc

IMPLEMENT_DYNCREATE(CMyWordDoc, CRichEditDoc)

BEGIN_MESSAGE_MAP(CMyWordDoc, CRichEditDoc)
    //|AFX_MSG_MAP(CMyWordDoc)
    // 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
    // Enable default OLE container implementation
    ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_LINKS,
        CRichEditDoc::OnUpdateEditLinksMenu)
    ON_COMMAND(ID_OLE_EDIT_LINKS, CRichEditDoc::OnEditLinks)
    ON_UPDATE_COMMAND_UI_RANGE(ID_OLE_VERB_FIRST,
        ID_OLE_VERB_LAST, CRichEditDoc::OnUpdateObjectVerbMenu)
END_MESSAGE_MAP()

////////////////////////////////////
// CMyWordDoc construction/destruction

CMyWordDoc::CMyWordDoc()
{
}

```

```

CMyWordDoc::~CMyWordDoc()
{
}

BOOL CMyWordDoc::OnNewDocument()
{
    if (!CRichEditDoc::OnNewDocument())
        return FALSE;
    return TRUE;
}

CRichEditCtrlItem* CMyWordDoc::CreateClientItem(KOBJECL * pobj) const
{
    return new CMyWordCtrlItem(pobj, (CMyWordDoc*) this);
}

////////////////////////////////////
// CMyWordDoc serialization

void CMyWordDoc::Serialize(CArchive& ar)
{
    CRichEditDoc::Serialize(ar);
}

////////////////////////////////////
// CMyWordDoc diagnostics

#ifdef _DEBUG
void CMyWordDoc::AssertValid() const
{
    CRichEditDoc::AssertValid();
}

void CMyWordDoc::Dump(CDumpContext& dc) const
{
    CRichEditDoc::Dump(dc);
}

#endif // _DEBUG

////////////////////////////////////
// CMyWordDoc commands

```

MyWordView.h

```

// MyWordView.h: interface of the CMyWordView class
//
////////////////////////////////////

#ifndef __AFX_MYWORDVIEW_H__C85C908D-A154-11D2-9E53-006008A82731__INCLUDED_
    #if !defined(
        AFX_MYWORDVIEW_H__C85C908D-A154-11D2-9E53-006008A82731__INCLUDED_
    )

```

```

#define AFX_MYWORDVIEW_H C85C908D_A154_11D2_8B53_006008A82731_ INCLUDED_

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

class CMyWordCtrlItem;

class CMyWordView : public CRichEditView
{
protected: // create from serialization only
    CMyWordView();
    DECLARE_DYNCREATE(CMyWordView)

// Attributes
public:
    CMyWordDoc * GetDocument();

// Operations
public:
    void GetFontInfo (LPTSTR pszFaceName, int& nSize);
    void ChangeFont (LPTSTR pszFaceName);
    void ChangeFontSize (int nSize);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMyWordView)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void OnInitialUpdate(); // called first time after construct
    //{{AFX_VIRTUAL
// Implementation
public:
    virtual ~CMyWordView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CMyWordView)
    afx_msg void OnDestroy();
    afx_msg void OnCharBold();
    afx_msg void OnCharItalic();

```

```

    afx_msg void OnCharUnderline();
    afx_msg void OnParaLeft();
    afx_msg void OnParaCenter();
    afx_msg void OnParaRight();
    afx_msg void OnUpdateCharBold(CCmdUI * pCmdUI);
    afx_msg void OnUpdateCharItalic(CCmdUI * pCmdUI);
    afx_msg void OnUpdateCharUnderline(CCmdUI * pCmdUI);
    afx_msg void OnUpdateParaLeft(CCmdUI * pCmdUI);
    afx_msg void OnUpdateParaCenter(CCmdUI * pCmdUI);
    afx_msg void OnUpdateParaRight(CCmdUI * pCmdUI);
    ///AFX_MSG
    afx_msg void OnUpdateLineNumber(CCmdUI * pCmdUI);
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in MyWordView.cpp
inline CMyWordDoc * CMyWordView::GetDocument()
{ return (CMyWordDoc *)m_pDocument; }
#endif

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

#endif
// !defined(
//     AFX_MYWORDVIEW_H__C85C508D-A1A4-11D2-8E53-006008A82731    INCLUDED_)

```

MyWordView.cpp

```

// MyWordView.cpp : implementation of the CMyWordView class
//

#include "stdafx.h"
#include "MyWord.h"

#include "MyWordDoc.h"
#include "CntrItem.h"
#include "MyWordView.h"

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

```

```

////////////////////////////////////
// CMyWordView

IMPLEMENT_DYNCREATE(CMyWordView, CRichEditView)
BEGIN_MESSAGE_MAP(CMyWordView, CRichEditView)
    //||AFX_MSG_MAP(CMyWordView)
    ON_WM_DESTROY()
    ON_COMMAND(ID_CHAR_BOLD, OnCharBold)
    ON_COMMAND(ID_CHAR_ITALIC, OnCharItalic)
    ON_COMMAND(ID_CHAR_UNDERLINE, OnCharUnderline)
    ON_COMMAND(ID_PARA_LEFT, OnParaLeft)
    ON_COMMAND(ID_PARA_CENTER, OnParaCenter)
    ON_COMMAND(ID_PARA_RIGHT, OnParaRight)
    ON_UPDATE_COMMAND_UI(ID_CHAR_BOLD, OnUpdateCharBold)
    ON_UPDATE_COMMAND_UI(ID_CHAR_ITALIC, OnUpdateCharItalic)
    ON_UPDATE_COMMAND_UI(ID_CHAR_UNDERLINE, OnUpdateCharUnderline)
    ON_UPDATE_COMMAND_UI(ID_PARA_LEFT, OnUpdateParaLeft)
    ON_UPDATE_COMMAND_UI(ID_PARA_CENTER, OnUpdateParaCenter)
    ON_UPDATE_COMMAND_UI(ID_PARA_RIGHT, OnUpdateParaRight)
    //||AFX_MSG_MAP
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_LINE, OnUpdateLineNumber)
END_MESSAGE_MAP()

////////////////////////////////////
// CMyWordView construction/destruction

CMyWordView::CMyWordView()
{
}

CMyWordView::~CMyWordView()
{
}

BOOL CMyWordView::PreCreateWindow(CREATESTRUCT& cs)
{
    return CRichEditView::PreCreateWindow(cs);
}

void CMyWordView::OnInitialUpdate()
{
    CRichEditView::OnInitialUpdate();

    CHARFORMAT cf;
    cf.cbSize = sizeof(CHARFORMAT);
    cf.dwMask = CFM_BOLD|CFM_ITALIC|CFM_UNDERLINE |
        CFM_PROTECTED|CFM_STRIKEOUT|CFM_FACE|CFM_SIZE;
}

```

```

        cf.dwEffects = 0;
        cf.yHeight = 240; // 240 twips == 12 points
        ::lstrcpy (cf.szFaceName, _T("Times New Roman"));
        SetCharFormat (cf);
    }

void CMywordView::OnDestroy()
{
    // Deactivate the item on destruction; this is important
    // when a splitter view is being used.
    CRichEditView::OnDestroy();
    COleClientItem* pActiveItem = GetDocument() -> GetInPlaceActiveItem(this);
    if (pActiveItem != NULL && pActiveItem -> GetActiveView() == this)
    {
        pActiveItem -> Deactivate();
        ASSERT(GetDocument() -> GetInPlaceActiveItem(this) == NULL);
    }
}

////////////////////////////////////
// CMywordView diagnostics

#ifdef _DEBUG
void CMywordView::AssertValid() const
{
    CRichEditView::AssertValid();
}

void CMywordView::Dump(CDumpContext& dc) const
{
    CRichEditView::Dump(dc);
}

CMywordDoc* CMywordView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument -> IsKindOf(RUNTIME_CLASS(CMyWordDoc)));
    return (CMywordDoc*)m_pDocument;
}

#endif // _DEBUG

////////////////////////////////////
// CMywordView message handlers

void CMywordView::OnCharBold()

{
    CHARFORMAT cf;
    cf = GetCharFormatSelection();
    if (!(cf.dwMask & CFM_BOLD) || (cf.dwEffects & CFE_BOLD))

```



```

        cf.dwEffects = CFE_BOLD;
    else
        cf.dwEffects = 0;

    cf.dwMask = CFM_BOLD;
    SetCharFormat(cf);
}

void CMYWordView::OnCharItalic()
{
    CHARFORMAT cf;
    cf = GetCharFormatSelection();

    if (!(cf.dwMask & CFM_ITALIC) || !(cf.dwEffects & CFE_ITALIC))
        cf.dwEffects = CFE_ITALIC;
    else
        cf.dwEffects = 0;

    cf.dwMask = CFM_ITALIC;
    SetCharFormat(cf);
}

void CMYWordView::OnCharUnderline()
{
    CHARFORMAT cf;
    cf = GetCharFormatSelection();

    if (!(cf.dwMask & CFM_UNDERLINE) || !(cf.dwEffects & CFE_UNDERLINE))
        cf.dwEffects = CFE_UNDERLINE;
    else
        cf.dwEffects = 0;

    cf.dwMask = CFM_UNDERLINE;
    SetCharFormat(cf);
}

void CMYWordView::OnParaLeft()
{
    OnParaAlign(PFA_LEFT);
}

void CMYWordView::OnParaCenter()
{
    OnParaAlign(PFA_CENTER);
}

void CMYWordView::OnParaRight()
{
    OnParaAlign(PFA_RIGHT);
}

```

```

void CMYWordView::OnUpdateCharBold(CCmdUI * pCmdUI)
{
    OnUpdateCharEffect (pCmdUI, CFM_BOLD, CFE_BOLD);
}

void CMYWordView::OnUpdateCharItalic(CCmdUI * pCmdUI)
{
    OnUpdateCharEffect (pCmdUI, CFM_ITALIC, CFE_ITALIC);
}

void CMYWordView::OnUpdateCharUnderline(CCmdUI * pCmdUI)
{
    OnUpdateCharEffect (pCmdUI, CFM_UNDERLINE, CFE_UNDERLINE);
}

void CMYWordView::OnUpdateParaLeft(CCmdUI * pCmdUI)
{
    OnUpdateParaAlign (pCmdUI, PFA_LEFT);
}

void CMYWordView::OnUpdateParaCenter(CCmdUI * pCmdUI)
{
    OnUpdateParaAlign (pCmdUI, PFA_CENTER);
}

void CMYWordView::OnUpdateParaRight(CCmdUI * pCmdUI)
{
    OnUpdateParaAlign (pCmdUI, PFA_RIGHT);
}

void CMYWordView::OnUpdateLineNumber(CCmdUI * pCmdUI)
{
    int nLine = GetRichEditCtrl().LineFromChar (-1) + 1;

    CString string;
    string.Format ( T("Line %d"), nLine);
    pCmdUI->Enable (TRUE);
    pCmdUI->SetText (string);
}

void CMYWordView::ChangeFont(LPCTSTR pszFaceName)
{
    CHARFORMAT cf;
    cf.cbSize = sizeof (CHARFORMAT);
    cf.bwMask = CFM_FACE;
    memcpy (cf.szFaceName, pszFaceName);
    SetCharFormat (cf);
}

```

```

void CMyWordView::ChangeFontSize(int nSize)
{
    CHARFORMAT cf;
    cf.cbSize = sizeof (CHARFORMAT);
    cf.dwMask = CFM_SIZE;
    cf.yHeight = nSize;
    SetCharFormat (cf);
}

void CMyWordView::GetFontInfo(LPTSTR pszFaceName, int& nSize)
{
    CHARFORMAT cf = GetCharFormatSelection ();
    ::lstrcpy (pszFaceName,
        cf.dwMask & CFM_FACE ? cf.szFaceName : _T (""));
    nSize = cf.dwMask & CFM_SIZE ? cf.yHeight : -1;
}

```

StyleBar.h

```

#ifndef __AFX_STYLEBAR_H__C85C9099_A154_11D2_8E53_006008A82731__INCLUDED_
#define __AFX_STYLEBAR_H__C85C9099_A154_11D2_8E53_006008A82731__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// StyleBar.h: header file
//

////////////////////////////////////
// CStyleBar command target

class CStyleBar : public CToolBar
{
// Attributes
public:
// Operations
public:
    static int CALLBACK EnumFontNameProc (ENUMLOGFONT * lpelf,
        NEWTEXTMETRIC * lpntm, int nFontType, LPARAM lParam);

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CStyleBar)
//{{AFX_VIRTUAL:
    virtual void OnUpdateCmdUI (CFrameWnd * pTarget,
        BOOL bDisableIfNoHndler);

```

```

// Implementation
protected:
    void InitTypefaceList (CDC* pDC);
    CFont m_font;
    CComboBox m_wndFontNames;
    CComboBox m_wndFontSizes;
    // Generated message map functions
    //{{AFX_MSG(CStyleBar)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    afx_msg void OnSelectFont ();
    afx_msg void OnSelectSize ();
    afx_msg void OnCloseUp ();
    DECLARE_MESSAGE_MAP()

.;

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

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

#endif
// !defined(
//     AFX_STYLBAR_H_ C85C9099_A154_11D2_8E53_006008A82731__INCLUDED_ )

```

StyleBar.cpp

```

// StyleBar.cpp : implementation file
//

#include "stdafx.h"
#include "MyWord.h"
#include "MyWordDoc.h"
#include "MyWordView.h"
#include "StyleBar.h"

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

////////////////////////////////////
// CStyleBar

BEGIN_MESSAGE_MAP(CStyleBar, CToolBar)

```

```

//||AFX_MSG_MAP(CStyleBar)
ON_WM_CREATE()
//||AFX_MSG_MAP
ON_CBN_SELENDOK(IDC_FONTNAMES, OnSelectFont)
ON_CBN_SELENDOK(IDC_FONTSIZES, OnSelectSize)
ON_CBN_CLOSEUP(IDC_FONTNAMES, OnCloseUp)
ON_CBN_CLOSEUP(IDC_FONTSIZES, OnCloseUp)
END_MESSAGE_MAP()

////////////////////////////////////
// CStyleBar message handlers

int CStyleBar::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    static int nFontSizes[] = {
        8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 26, 28, 32, 36, 48, 72
    };

    if (CToolBar::OnCreate(lpCreateStruct) == -1)
        return -1;

    //
    // Load the toolbar.
    //
    if (!LoadToolBar(IDR_STYLE_BAR))
        return -1;

    //
    // Create an 8-point MS Sans Serif font for the combo boxes.
    //
    CClientDC dc(this);
    m_font.CreatePointFont(80, _T("MS Sans Serif"));
    CFont* pOldFont = dc.SelectObject(&m_font);

    TEXTMETRIC tm;
    dc.GetTextMetrics(&tm);
    int cxChar = tm.tmAveCharWidth;
    int cyChar = tm.tmHeight + tm.tmExternalLeading;

    dc.SelectObject(pOldFont);

    //
    // Add the font name combo box to the toolbar.
    //
    SetButtonInfo(8, IDC_FONTNAMES, TBBS_SEPARATOR, cxChar * 32);

    CRect rect;

```

```

    GetItemRect (8, &rect);
    rect.bottom = rect.top + (cyChar * 16);

    if (!m_wndFontNames.Create (WS_CHILD|WS_VISIBLE|WS_VSCROLL |
        CBS_DROPDOWNLIST|CBS_SORT, rect, this, IDC_FONTNAMES))
        return -1;

    m_wndFontNames.SetFont (&m_font);
    InitTypefaceList (&dc);

    //
    // Add the font size combo box to the toolbar.
    //
    SetButtonInfo (10, IDC_FONTSIZES, TBBS_SEPARATOR, cxChar * 12);

    GetItemRect (10, &rect);
    rect.bottom = rect.top + (cyChar * 14);

    if (!m_wndFontSizes.Create (WS_CHILD|WS_VISIBLE|WS_VSCROLL |
        CBS_DROPDOWNLIST, rect, this, IDC_FONTSIZES))
        return -1;

    m_wndFontSizes.SetFont (&m_font);

    CString string;
    int nCount = sizeof (nFontSizes) / sizeof (int);
    for (int i = 0; i < nCount; i++) {
        string.Format (_T ("%d"), nFontSizes[i]);
        m_wndFontSizes.AddString (string);
    }

    return 0;
}

void CStyleBar::OnSelectFont ()
{
    TCHAR szFaceName[LF_FACESIZE];
    int nIndex = m_wndFontNames.GetCaretIndex ();
    m_wndFontNames.GetLBText (nIndex, szFaceName);

    CMyWordView * pView
        = (CMyWordView *) ((CFrameWnd *) AfxGetMainWnd ()) -> GetActiveView ();
    pView -> ChangeFont (szFaceName);
}

void CStyleBar::OnSelectSize ()
{
    TCHAR szSize[8];

```

```

        int nIndex = m_wndFontSizes.GetCurSel();
        m_wndFontSizes.GetLBText(nIndex, szSize);

        int nSize = atoi(szSize) * 20; // Need twips

        CMyWordView* pView =
            (CMyWordView*) ((CFrameWnd*) AfxGetMainWnd()) -> GetActiveView();
        pView -> ChangeFontSize(nSize);
    }

    void CStyleBar::OnCloseUp()
    {
        ((CFrameWnd*) AfxGetMainWnd()) -> GetActiveView() -> SetFocus();
    }

    void CStyleBar::InitTypefaceList(CDC* pDC)
    {
        ::EnumFontFamilies(pDC -> m_hDC, NULL,
            (FONTENUMPROC) EnumFontNameProc, (LPARAM) this);
    }

    int CALLBACK CStyleBar::EnumFontNameProc(ENUMLOGFONT* lpelf,
        NEWTEXTMETRIC* lpntm, int nFontType, LPARAM lParam)
    {
        CStyleBar* pWnd = (CStyleBar*) lParam;
        if (nFontType & TRUETYPE_FONTTYPE)
            pWnd -> m_wndFontNames.AddString(lpelf -> elfLogFont.lfFaceName);
        return 1;
    }

    void CStyleBar::OnUpdateCmdUI(CFrameWnd* pTarget, BOOL bDisableIfNoHandler)
    {
        CToolBar::OnUpdateCmdUI(pTarget, bDisableIfNoHandler);

        CWnd* pWnd = GetFocus();
        if ((pWnd == &m_wndFontNames) || (pWnd == &m_wndFontSizes))
            return;

        //
        // Get the font name and size.
        //
        int nTwips;
        TCHAR szFaceName[LF_FACESIZE];

        CMyWordView* pView =
            (CMyWordView*) ((CFrameWnd*) AfxGetMainWnd()) -> GetActiveView();
        pView -> GetFontInfo(szFaceName, nTwips);
    }

```

```

//
// Update the font name combo box.
//
TCHAR szSelection[16_FACESIZE];
m_wndFontNames.GetWindowText (szSelection,
    sizeof (szSelection) / sizeof (TCHAR));

if (!lstrcmp (szFaceName, szSelection) != 0) :
    if (szFaceName[0] == 0)
        m_wndFontNames.SetCurSel (-1);
    else {
        if (m_wndFontNames.SelectString (-1, szFaceName) == CB_ERR)
            m_wndFontNames.SetCurSel (-1);
        !
    }

//
// Update the font size combo box.
//
TCHAR szSize[4];
m_wndFontSizes.GetWindowText (szSize,
    sizeof (szSize) / sizeof (TCHAR));
int nSizeFromComboBox = atoi (szSize);
int nSizeFromView = mTwips / 20;
if (nSizeFromComboBox != nSizeFromView) {
    if (mTwips -- > 0)
        m_wndFontSizes.SetCurSel (-1);
    else
        CString string;
        string.Format ( _T ("%d"), nSizeFromView);
        if (m_wndFontSizes.SelectString (-1, string) == CB_ERR)
            m_wndFontSizes.SetCurSel (-1);
    !
}
}

```

图 12-9 MyWord 应用程序

12.3.1 主工具栏

MyWord 的主工具栏是一个标准 CToolBar，它同样式栏和状态栏一起在 CMainFrame::OnCreate 中被创建。主工具栏创建后，则添加样式 CBRS_TOOLTIPS、CBRS_FLYBY 和 CBRS_

SIZE_DYNAMIC,并通过 CBRSS_ALIGN_ANY 参数调用 CToolBar::EnableDocking,使工具栏可以被停放在框架窗口的任一侧。调用 DockControlBar 把工具栏停放在窗口顶部的默认位置上,使它能脱离窗口并浮动起来。如果应用程序曾运行过,则在 CMainFrame::OnCreate 中调用 LoadBarState 可以恢复工具栏以前的位置。

主工具栏中所有按钮的处理程序,即 MyWord 菜单中所有菜单项的处理程序,都是主框架提供的。照例,CWinApp 提供 File 菜单中 New、Open 和 Exit 命令的处理程序,而 CDocument 处理 Save 和 Save As 命令。CRichEditView 提供 Edit 菜单中菜单项命令的处理程序(当然,所有关系都已在消息映射表中),而 CFrameWnd 处理 View 菜单中的命令。CRichEditView 还提供所有 Edit 命令的更新函数程序。该函数对工具栏中 Cut、Copy、Paste 和 Undo 按钮根据用户的操作在有效和失效间自动切换进行了解释。如果想弄明白,可以敲入一两行文字,并加亮显示几个字符表示选中。当第一个字符被选中时,Cut 和 Copy 按钮会变亮;当选中被取消时,这两个按钮又会变灰。因为 CRichEditView 消息映射表中有下列几项,所以更新都是自动的:

```
ON_UPDATE_COMMAND_UI (ID_EDIT_CUT, OnUpdateNeedSel)
ON_UPDATE_COMMAND_UI (ID_EDIT_COPY, OnUpdateNeedSel)
```

细查 MFC 源代码文件 Viewrich.cpp 中的 CRichEditView 消息映射表,您可以看到 CRichEditView 为其提供了默认命令和更新处理程序的所有命令。

12.3.2 样式栏

MyWord 样式栏是 CToolBar 的派生类 CStyleBar 的一个实例。样式栏在构造框架窗口时构造,并在 CMainFrame::OnCreate 中创建。但它还包含自己的 OnCreate 函数,该函数创建和初始化字体名和字体大小组合框。其他的 CStyleBar 成员函数包括:OnSelectFont,它应用在字体名组合框中选择的字体;OnSelectSize,它的主要参数是字体大小组合框中选择的尺寸;OnCloseUp,当任何一个组合框的下拉列表框被关闭时,它将输入焦点恢复到视图中;InitTypefaceList 和 EnumFontNameProc,它们互相配合,枚举字体并在字体名组合框中加入它们的名字;以及 OnUpdateCmdUI,它更新组合框,使样式栏显示的字体名和字体大小与插入符处的字符和选中字符保持一致。

MyWord 视图类提供样式栏按钮的命令和更新处理程序。例如:单击 Bold 按钮、激活 CMyWordView::OnCharBold。该函数是这样实现的:

```
void CMyWordView::OnCharBold ()
{
    CHARFORMAT cf;
    cf = GetCharFormatSelection();
    if (!(cf.dwMask & CFM_BOLD) || !(cf.dwEffects & CFE_BOLD))
```

```

        cf.dwEffects = CFE_BOLD;
    e se
        cf.dwEffects = 0;

    cf.dwMask = CFM_BOLD;
    SetCharFormat (cf);
}

```

GetCharFormatSelection 是一个 CRichEditView 函数,它返回 CHARFORMAT 结构,其中包含视图中当前选中文本的信息,如果没有选中文本,则包含默认字符格式。SetCharFormat 是另一个 CRichEditView 函数,它将 CHARFORMAT 结构描述的属性应用于选中文本。如果当前没有文本被选中,SetCharFormat 则设置视图的默认字符格式。

通过在 CHARFORMAT 结构的 dwMask 域设置 CFM_BOLD 位,并在 dwEffects 域设置或消除 CFE_BOLD 位,可以切换黑体字文本。为了确定 CFE_BOLD 标志的适当设置,OnCharBold 检查 CHARFORMAT 结构(由 GetCharFormatSelection 返回)中 CFM_BOLD 和 CFE_BOLD 标志。如果当前选中文本中混合有黑体和非黑体文本,则 CFM_BOLD 标志是空的。如果 CFM_BOLD 有设定值,则选中文本可能是纯黑体或纯非黑体文本,或者当前没有选中文本。在这两种情况下,CFE_BOLD 标志都指明选中文本(或默认文本)是黑体还是非黑体。可以在 5 种可能的场合下调用 OnCharBold。表 12-3 描述了各种情况并给出了结果。视图的 OnCharItalic 和 OnCharUnderline 处理程序运用了类似逻辑切换斜体和下划线体。

表 12-3 调用 OnCharBold 的场合及结果

调用 OnCharBold 的场合	dwMask&CFM_BOLD	dwEffects&CFE_BOLD	OnCharBold 产生的动作
选中一个或多个字符; 选中文本中混合有黑体和非黑体	0	未定义	使选中文本中的所有字符为黑体
选中一个或多个字符; 选中文本为纯黑体正文	非零	非零	使选中文本中的所有字符为非黑体
选中一个或多个字符; 选中文本为纯非黑体正文	非零	0	使选中文本中的所有字符为黑体
没有文本选中;默认字符格式为黑体	非零	非零	将默认字符格式设为非黑体
没有文本选中;默认字符格式为非黑体	非零	0	将默认字符格式设为黑体

因为段落对齐按钮处理函数的动作不依赖于当前段落的对齐样式,所以这些处理函数就简单得多。CRichEditView 提供了一个方便的 OnParaAlign 函数,它将段落对齐设置成左对齐、右对齐和居中对齐(不幸的是:作为 CRichEditView 的基础,CRichEditView 和多行文本编

辑控件都不支持两端完全对齐的文本)。OnParaLeft 中的语句

```
OnParaAlign (PFA_LEFT);
```

选中左对齐文本。如果视图中没有文本被选中,则 OnParaAlign 重新调整含有插入符段落的格式。如果有文本选中,则所有选中段落的文本都被转换成左对齐格式。

样式栏中每个按钮都对应着一个处理函数,而这些处理函数或调用 CRichEditView::OnUpdateCharEffect,或调用 CRichEditView::OnUpdateParaAlign。除了恰当选中和取消选中这些按钮,在选中文本包含多种字符格式或段落对齐样式的组合时,这些 CRichEditView 函数还能把按钮设置成中间状态。可以试试看。如果还没有文本,先输入一些文本。然后加亮显示一些字符,单击 Italic 按钮使选中文本斜体化,然后选中一段字符,其中既包含斜体字符又包含非斜体字符。因为 OnUpdateCharItalic 调用 OnUpdateCharEffect,所以 Italic 按钮会变成浅灰色,表示选中文本包含多种字符格式。又因为每个样式栏按钮都有一个更新处理程序,所以尽管没有赋予它们 TBBS_CHECKBOX 或 TBBS_CHECKGROUP 样式,但它们的行为方式和复选按钮、单选按钮相似。

如果在组合框中选中一种字体或字体大小,样式栏提取字体名或字体大小,并调用视图类的公用成员函数来实现数据交换。选中一种字体会激活 CStyleBar::OnSelectFont,该函数通过 CMyWordView::ChangeFont 把新字体名传递给视图。接着,ChangeFont 通过在 CHARFORMAT 结构的 dwMask 域设置 CFM_FACE 标志,把字体名复制到该结构的 szFaceName 域,并调用 SetCharFormat,最终改变视图中的字体。

```
void CMyWordView::ChangeFont (LPCTSTR pszFaceName)
{
    CHARFORMAT cf;
    cf.cbSize = sizeof (CHARFORMAT);
    cf.dwMask = CFM_FACE;
    ::lstrcpy (cf.szFaceName, pszFaceName);
    SetCharFormat (cf);
}
```

CStyleBar::OnSelectSize 也以类似的步骤调用视图的 ChangeFontSize 成员函数,最终实现字体大小的改变。传递给 CRichEditViews 的字体大小是以缇(twip)表示的,一缇等于一个点的 1/20。因此,在调用 ChangeFontSize 之前,OnSelectSize 通过把从组合框获取的点的尺寸乘以 20,使点转换为缇。

这样就产生了一个问题:因为命令消息是在组合框中某项被选中时产生的,它服从命令传递路径,为什么 MyWord 不让视图直接处理组合框通知呢?实际上,这“或许是”理想情况,但也带来一个问题。因为组合框是样式栏类的保护型成员变量,所以视图可能无法从组合框中提取选中项。尽管通过把组合框和样式栏转变为框架窗口类的公用数据成员,可以解决这个矛盾,但是要明白:保护型数据成员会提供更严谨的封装。如果由样式栏响应组

合框消息,并把消息通过公用成员函数传递给视图,则样式栏可以隐藏它的数据,同时仍然将样式的变化传给视图。

插入符在文档中移动并实现选中时,如果要使组合框中的选中项和视图中的字符格式相匹配,则 CStyleBar 需重载它从 CToolBar 继承来的 OnUpdateCmdUI 函数,并根据视图中的信息更新组合框。如果两个组合框都没有输入焦点,则当下拉列表显示时如果调用 OnUpdateCmdUI,组合框就不会闪烁。在确信组合框均无输入焦点后,OnUpdateCmdUI 调用 CMyWordView::GetFontInfo 获取当前字体和大小。如果从视图获取的字体和字体组合框中的选中项不匹配,OnUpdateCmdUI 则改变组合框中的选中项。类似地,如果组合框的选中大小和 GetFontInfo 报告的大小不匹配,则组合框中的选中项被更新。如果选中项没有改变,则不再改动选中项,从而防止因反复(不必要)的更新而闪烁。如果 GetFontInfo 中的字体或字体大小和组合框中的任何一项都不匹配,或视图中被选中的文本包含多种字体或字体尺寸,更新处理程序都能成功地取消组合框选中项。

如果 MyWord 运行时安装的字体发生了变化,CStyleBar 则不会更新字体组合框中的字体列表。如果添加或删除了一些字体,Windows 则给所有顶层窗口发送 WM_FONTCHANGE 消息,告诉它们这个变化。在应用程序运行时,如果要响应可用字体类型的变化,则在框架窗口的消息映射表添加 ON_WM_FONTCHANGE 项,以及相应的 OnFontChange 处理程序。该消息映射表项和处理程序必须是框架窗口类的成员。因为 WM_FONTCHANGE 消息不传递,而命令消息则传递。

为了简化字体大小组合框中更新选中项的逻辑,MyWord 的样式栏只列有 TrueType 字体。如果字体名组合框还包含光栅字体,因为光栅字体只有有限的几种尺寸,则字体名组合框中选中项每次变化时,都要重新初始化字体大小组合框。如果将用户的字体选择限制在 TrueType,则列在字体大小组合框中的点的大小和字体名组合框中的字体选项无关,因为 TrueType 字体可以精确给出从 1 到 999 点间的任意尺寸。

12.3.3 再谈 CRichEditView

MyWord 的大部分功能来自于 CRichEditView。CRichEditView 的核心是公用控件库提供的强大的多行文本编辑控件。MFC 的 CRichEditView 类在封装多行文本编辑控件功能时也不是独立工作的,CRichEditDoc 和 CRichEditCtrlItem 也参与了部分工作。CRichEditDoc 代表保存在 CRichEditView 中的数据。CRichEditView 可以包含链接和嵌入的 OLE 对象,而 CRichEditCtrlItem 则代表这些 OLE 对象。在从 CRichEditView 派生视图类时,您还必须从 CRichEditDoc 派生一个文档类,并重载虚函数 CRichEditDoc::CreateClientItem。MyWord 的 CMyWordDoc 文档类通过创建一个 CRichEditCtrlItem 对象并返回一个指针实现 CreateClientItem:

```
CRichEditCtrlItem* CMyWordDoc::CreateClientItem(REOBJECT* preo) const
```

```

    }
    return new CMYWordCtrlItem(pres, (CMYWordDoc*) this);
}

```

这样简单的重载可以使 Edit 菜单中的 Paste 和 Paste Special 命令将 OLE 项粘贴到文档中。可以示范一下,现在复制 Windows Paint 小应用程序下创建的图片,并把它粘贴到 MyWord 文档中。然后双击 MyWord 中的嵌入对象,这时,Paint 应用程序的菜单和工具栏会同 MyWord 的菜单和工具栏同时出现在屏幕上,使您能在现场编辑图片。如果保存了文档,则嵌入的 Paint 对象也被保存了。下次加载文档时,它就会以这次文档关闭时被保存的形式出现。

可能您还没注意到,MyWord 完全能够保存您创建的文档并把它们再加载进来。它甚至能读取由其他文字处理程序创建的 RTF 文件并序列化 OLE 对象。CMYWordDoc::Serialize 只包含一个语句:

```
CRichEditDoc::Serialize(ar);
```

因为 CRichEditDoc 自己能处理序列化,所以在 CMYWordDoc 里您找不到任何实现序列化的代码。CRichEditDoc::Serialize 通过调用视图的 Serialize 函数将数据输入或输出 CRichEditView。而 Serialize 函数又依赖多行文本编辑控件的传输能力(欲知详细情况,请参见 EM_STREAMIN 和 EM_STREAMOUT 消息的文档说明。这两个消息可以被发送到多行文本编辑控件和与之等价的 StreamIn 和 StreamOut 函数,MFC CRichEditCtrl 类的成员)。编写一个 SDK 应用程序,使它能在多行文本编辑控件中保存和加载文档。这个工作比较容易。但在 MFC 中这个工作就极其简单了。因为 CRichEditDoc 和 CRichEditView 同主框架的其他组成部分一起可以替您处理整个序列化过程。

在默认方式下,CRichEditDoc 以多行文本格式序列化文档。通过在保存文档前将 CRichEditDoc 的数据成员 m_bRTF 设置为 FALSE,可以指示 CRichEditDoc 写缺少格式化信息和 OLE 对象的文本文件。因此,通过在打开文档前将 m_bRTF 设置成 FALSE,您可以以纯文本格式读取文件。除 RTF 文件外,要使 MyWord 也能读写文本文件并不困难;但是必须在并行化的前端添加一些逻辑判断,确定待读文件的类型。如果 m_bRTF 为 TRUE,则 CRichEditDoc 不能加载文本文件;如果 m_bRTF 为 FALSE,在读取 RTF 文档时,CRichEditDoc 将 RTF 格式化命令转换为普通文本。CRichEditDoc 序列化选项的全面介绍超出了本书范围,但是如果您对此有兴趣,可以先了解一下 MFC 提供的 Wordpad 源代码。

12.4 组合栏

Internet Explorer 3.0 给 Windows 引入了一个新的控件类型:组合栏控件。组合栏容纳其他控件。在组合栏中添加“控件”,可以充实组合栏控件;每个带形区可以包含一个子窗口,比如工具栏、按钮或组合框。您可以随意添加带形区。一旦组合栏显示,用户就可以通过移

动和调整带形区的大小将组合栏控件设置成他喜欢的样子。每个带形区都可以获取图像列表中的图像、标签(文本字符串)和相关的位图。如果设定了图像和标签,它们就会显示在带形区表面。如果设定了位图,则不管带形区中的显示内容,位图都会沿水平和竖直方向铺列下去,从而形成有风格的背景图案。还记得 Internet Explorer 3.0 中那个有纹理背景的工具栏吗?那个工具栏实际上是一个组合栏控件,其中包含着一个非常普通的工具栏控件。位图则提供了有纹理的背景。

MFC 6.0 引入了两个新类, CReBar 和 CReBarCtrl, 可以简化组合栏控件的编程过程。CReBarCtrl 是一个初级类, 它只给未成形的组合栏控件提供简单的配置。CReBar 则是一个高级类, 它使在 MFC 应用程序中添加组合栏就像添加工具栏与状态栏一样容易。CReBar 只公开 3 个成员函数:

- Create 用 CReBar 对象创建组合栏
- GetReBarCtrl 返回一个组合栏控件的 CReBarCtrl 引用。
- AddBar 在组合栏中添加控件。

有了 CReBar 和它的成员函数的帮助, 创建组合栏型工具栏(如 Visual C++ 中用到的)就是极简单的事了。下面的示例将 MFC 工具栏转换为组合栏。它首先创建 CToolBar, 而后使它成为 CReBar 中的一个控件。

```
m_wndToolBar.CreateEx(this);
m_wndToolBar.LoadToolBar(IDR_TOOLBAR);
m_wndReBar.Create(this);
m_wndReBar.AddBar(&m_wndToolBar);
```

可以使用 AddBar 的第二和第三个可选参数指定标签和背景位图。例如: 如果 m_bitmap 是一个 CBitmap 对象, 代码

```
m_bitmap.LoadBitmap(IDB_BKGND);
m_wndToolBar.CreateEx(this, TBSTYLE_FLAT|TBSTYLE_TRANSPARENT);
m_wndToolBar.LoadToolBar(IDR_TOOLBAR);
m_wndReBar.Create(this);
m_wndReBar.AddBar(&m_wndToolBar, _T("Main"), &m_bitmap);
```

赋给工具栏标签 Main, 并用 ID 为 IDB_BKGND 的位图资源作工具栏的背景。如果以这种方式使用背景位图, 则一定要定义工具栏样式为 TBSTYLE_FLAT 和 TBSTYLE_TRANSPARENT, 并用浅灰作工具栏按钮的背景色。否则, 按钮背景就会画在背景位图顶部上, 从而不能得到预期的效果。

如果在 AppWizard 的 Step 4 对话框中选中 Internet Explorer ReBars 框(如图 12-10 所示), AppWizard 就会在它生成的工具栏上套一个组合栏。如果您还想在组合栏上做些工作, 比

如,添加标签或位图,或者,如果应用程序有多个工具栏,要求每个都放在组合栏中,那么您必须自己添加代码。

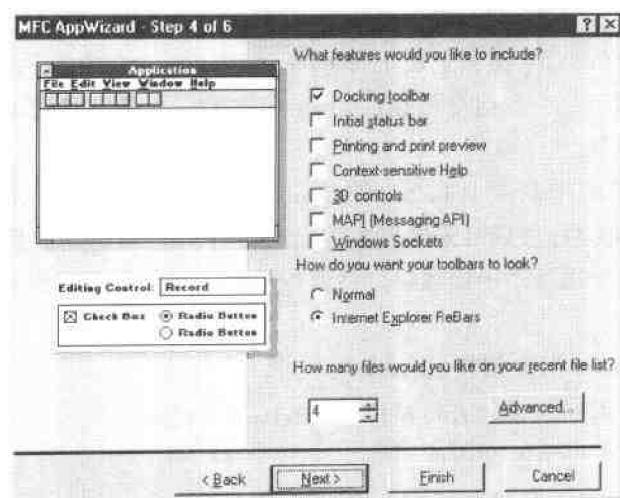


图 12-10 使用 AppWizard 用控件栏套装工具栏

第 13 章 打印和打印预览

对于初学者,学习打印是 Microsoft Windows 程序设计中非常难的一部分。打印是由 GDI 来实现的。一方面 GDI 为各种打印机提供了一个与设备独立的接口,这大大简化了打印工作;另一方面,GDI 大大丰富了文档的输出手段。用于屏幕的 GDI 函数可以用来输出到打印机,但需提供一些相关信息,如给出打印页码、结束未完成的打印工作等。同时一个好的有竞争力的软件,还需提供打印预览功能。

在第 9 章到第 12 章中,您已经看到了文档/视图体系结构是如何简化单文档界面(SDI)和多文档界面(MDI)应用程序开发的,MFC 完成了大部分工作。在本章中,将要学习如何用文档/视图结构来简化打印和打印预览。即使用 MFC 来实现打印功能也不轻松,不过要感谢主框架提供的支持,那些乏味的编写和测试将文档提交给打印机和其他硬拷贝设备的程序代码的工作大大减少了。一旦让应用程序具备打印功能,打印预览功能自然就有了。

13.1 通过文档和视图打印

MFC 的打印体系结构是以 GDI 打印函数和 CView 的虚函数为核心发展起来的。想理解外在的功能,最好先了解内部的结构。本章中按下列步骤学习实现 MFC 应用程序支持打印和打印预览的方法:

- 学习 Windows 打印模型,并了解基于 SDK 的应用程序打印文档的过程。
- 学习 Windows 体系结构和 MFC 打印体系结构间的关系,以及 MFC 应用程序的打印机制。
- 开发一个纯打印程序,说明同样的程序代码既可以用于屏幕输出,也可以用于打印机输出。
- 开发一个功能更强的打印程序,它的打印和打印预览功能与商业应用程序中的相同。

您会发现,MFC 打印输出除了必须编页码以外与屏幕输出差别不大。由于 MFC 几乎处理了所有工作,在编写打印程序时仅需安排页的分隔和在打印纸上的输出位置。

13.1.1 Windows 打印体系结构

如果不利用 MFC,由 Windows 应用程序打印一个文档会涉及许多步骤。首先要获取与

打印相关的设备描述表。如同应用程序需要屏幕设备描述表给屏幕发送输出一样,它也需要打印机设备描述表给打印机发送输出。如果已知所用打印机的设备名称,可以调用 Win32::CreateDC 函数或 MFC 的 CDC::CreateDC 来创建设备描述表:

```
CDC dc;
dc.CreateDC (TEXT ("HP LaserJet 4IP"), NULL, NULL);
```

如果不知道设备名称而希望应用程序输出到默认的打印机上,则可以使用 MFC 中的 CPrintDialog::GetDefaults 和 CPrintDialog::GetPrinterDC 函数来创建设备描述表:

```
CDC dc;
CPrintDialog dlg (FALSE);
dlg.GetDefaults ();
dc.Attach (dlg.GetPrinterDC ());
```

如果想让用户选择打印机,则可以使用 CPrintDialog::DoModal 显示一个“打印”对话框(一个操作系统提供的常用对话框)并在对话框关闭以后调用 CPrintDialog::GetPrinterDC 来获得 DC:

```
CDC dc;
CPrintDialog dlg (FALSE);
if (dlg.DoModal () == IDOK)
    dc.Attach (dlg.GetPrinterDC ());
```

为防止耗尽资源,应该在不需要时删除用这些方法获得的打印机 DC 来释放资源。如果附加 DC 的 CDC 对象是在堆栈上创建的,删除会自动执行。

一旦获得了打印机 DC,就可以准备打印了。下一步是调用::StartDoc 或 MFC 中等价的 CDC::StartDoc 来标记打印工作的开始。CDC::StartDoc 仅接受一个参数:指向 DOCINFO 结构的指针,该结构包含了文档的描述性名称、(如果打印输出到文件中而不是打印机上时)目标文件名称以及其他与打印工作有关的信息。语句

```
DOCINFO di;
::ZeroMemory (&di, sizeof (DOCINFO));
di.cbSize = sizeof (DOCINFO);
di.lpszDocName = TEXT ("Budget Figures for the Current Fiscal Year");
dc.StartDoc (&di);
```

在与 CDC 对象 dc 关联的打印机上开始打印工作。如果正在打印文档时打开一个打印机窗口,字符串“Budget Figures for the Current Fiscal Year”会标识打印工作。如果 StartDoc 失败,则返回 0 或小于 0 的值。如果成功就返回一个等于打印工作 ID 的正整数。可以与 Win32 打印控制函数如::GetJob 和 ::SetJob 一起使用打印工作 ID。

接下来就是在打印纸上输出。文本和图形是用 GDI 函数提交给打印机的。如果 dc 指的是屏幕 DC,语句

```
dc.Ellipse (0, 0, 100, 100);
```

将在屏幕上画一个 100 逻辑单位宽 100 逻辑单位高的椭圆。如果 dc 指的是打印机设备描述表,圆就会被画在打印机上。输出页是调用 CDC::StartPage 和 CDC::EndPage 制定的,两个函数分别表明每页的开始和结束。一个包含 nPageCount 输出页的文档可以如下方式打印:

```
for (int i = 1; i <= nPageCount; i++) {
    dc.StartPage();
    // Print page i
    dc.EndPage();
}
```

可以简单地认为,调用 EndPage 类似于给打印机输出表单送纸字符。在 StartPage 和 EndPage 之间要调用 CDC 成员函数来打印页中内容。即使文档只包含一页内容也应该调用 StartPage 和 EndPage。

在第一次编写打印程序代码时程序员常犯的错误是没有给每一页都初始化打印机 DC。在 Windows 95 和 Windows 98 中,设备描述表的默认属性在每次调用 StartPage 时都被恢复。不能在创建 DC 并选择了字体或设置了映射模式之后,希望这些属性像屏幕 DC 那样能永远保留下去,相反,每一页都必须重新初始化(在 Microsoft Windows NT 3.5 和以后版本中,打印机 DC 在 StartPage 和 EndPage 整个调用中保持其设置,但即使是 Windows NT 应用程序,如果它在 Windows 95 和 Windows 98 中运行,也要在每页开头重新初始化设备描述表),例如:使用 MM_LOENGLISH 映射模式打印时,应该在每页的开头处调用 CDC::SetMapMode,如下:

```
for (int i = 1; i <= nPageCount; i++) {
    dc.StartPage();
    dc.SetMapMode (MM_LOENGLISH);
    // Print page i.
    dc.EndPage();
}
```

下列代码则以默认的 MM_TEXT 映射模式打印。

```
dc.SetMapMode (MM_LOENGLISH);
for (int i = 1; i <= nPageCount; i++) {
    dc.StartPage();
    // Print page i.
    dc.EndPage();
}
```

在打印完最后一页以后,应用程序通过调用 CDC::EndDoc 来结束打印工作。打印工作

在进行以下处理时有些复杂,如果前一次调用 `EndPage` 返回的代码说明打印工作已经被 GDI 终止,这时就不应该再调用 `EndDoc` 了。`EndPage` 的返回值大于零,说明页内容已成功地输出到打印机。为零或负数说明要么出现了错误,要么是打印页在打印时被用户取消了打印工作。在两种情况下,返回代码都会等于表 13-1 中的一个。

表 13-1 返回代码值

返回代码	说 明
<code>SP_ERROR</code>	由于未指出的原因打印工作中途失败
<code>SP_APPABORT</code>	由于用户在显示打印工作状态的对话框中单击了“取消”按钮,打印工作被中途终止
<code>SP_USERABORT</code>	由于用户通过操作系统内部命令取消了打印,打印工作被中途终止
<code>SP_OUTOFDISK</code>	系统缺少磁盘空间,以后的打印数据不能在假脱机状态下输入输出
<code>SP_OUTOFMEMORY</code>	系统缺少内存空间,以后的打印数据不能在假脱机状态下输入输出

下列循环语句将打印文档的每一页,并且仅当每页得到成功打印后才在打印工作结束时调用 `EndDoc` :

```
if (dc.StartDoc (&di) > 0) {
    BOOL bContinue = TRUE;

    for (int i=1; i<=nPageCount && bContinue; i++) {
        dc.StartPage ();
        // Initialize the device context.
        // Print page i.
        if (dc.EndPage () <= 0)
            bContinue = FALSE;
    }

    if (bContinue)
        dc.EndDoc ();
    else
        dc.AbortDoc ();
}
```

正像 `EndDoc` 标志着打印成功结束一样, `CDC::AbortDoc` 标志着未完成打印工作的结束。也可以在 `StartPage` 和 `EndPage` 之间调用 `AbortDoc` 来在最后一页打印完之前终止打印工作。

中止处理和中止对话框

如果在 Windows 下给打印机输出结果只有这么多工作的话,打印也就不会成为可怕的任务了。一个非常巨大的打印工作可能会花几分钟或几小时来完成,因此用户应该有能力在工作完成以前结束打印。Windows 应用程序习惯上通过显示一个包含“取消”按钮的对话框

框来给用户提供一个取消打印工作的方法。单击“取消”按钮会促使 EndPage 返回 SP_APPABORT 来取消打印。将“取消”按钮与打印程序代码联系起来的机制是 Windows 称为中止处理的函数。

中止处理是导出的回调函数, Windows 在处理打印输出时会反复调用它。它的原型如下:

```
BOOL CALLBACK AbortProc (HDC hDC, int nCode)
```

hDC 保存打印机设备描述表的句柄。nCode 在打印正常进行时为零, 如果打印假脱机程序临时缺少磁盘空间时为 SP_OUTOFDISK。nCode 通常被忽略, 这是因为打印假脱机程序通过等待更多的磁盘空间被释放来响应 SP_OUTOFDISK 条件。中止处理要作如下两项工作:

- 用 ::PeekMessage 检查消息队列, 并检索和调度任何等待着的消息
- 告诉 Windows 是否继续打印, 返回 TRUE 继续打印, 返回 FALSE 中止

通过调用 ::SetAbortProc 或 CDC::SetAbortProc 给 Windows 传递中止处理函数的地址。一个非常简单的中止处理如下:

```
BOOL CALLBACK AbortProc (HDC hDC, int nCode)
{
    MSG msg;
    while (::PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
        AfxGetThread ()->PumpMessage ();
    return TRUE;
}
```

在 AbortProc 中的消息循环允许当单击打印状态对话框中的“取消”按钮时, 产生的 WM_COMMAND 消息即使在应用程序忙于打印时都可以被传递给窗口处理程序。在 16 位 Windows 中, 消息循环在多任务处理中起着重要的作用, 通过控制转换使得打印假脱机程序和运行于系统中的其他处理过程都可以得到 CPU 时间。在 Windows 95 和 Windows 98 中, 中止处理中的控制转换在 32 位应用程序输出到 16 位打印机驱动程序时, 通过减少对 Win16Mutex (内部标志, 用来在 16 位应用程序在 16 位内核中执行操作时封锁 32 位应用程序对 16 位内核的使用) 的争用增强了多任务操作的功能。

在打印开始前 (调用 StartDoc 之前), 应用程序将调用 SetAbortProc 来设置中止处理, 通过 FALSE 参数调用 CWnd::EnableWindow 来使自己的窗口失效, 并显示打印状态对话框或“中止”对话框 (一个非模态对话框, 其中包含“取消”按钮和一个以上静态控件, 列出了文档的文件名和当前打印的总页数)。主窗口失效可以确保不会有其他输入打断打印设置过程。在打印设置完成和对话框关闭后窗口恢复有效。如果单击的是“取消”按钮, 对话框会将标志 bUserAbort 由 FALSE 设置为 TRUE。如果 bUserAbort 为 TRUE, 中止处理函数也会返回 FALSE 来关闭打印。

```

BOOL CALLBACK AbortProc (HDC hDC, int nCode)
{
    MSG msg;
    while (!bUserAbort &&
           ::PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
        AfxGetThread() > PumpMessage();
    return !bUserAbort;
}

```

这样,如果单击的不是“取消”按钮,AbortProc 总是返回非零值,打印会无阻碍地进行下去。如果单击的是“取消”按钮,bUserAbort 就会由 FALSE 变为 TRUE,下次对 AbortProc 的调用返回 0,Windows 就终止打印过程。EndPage 将返回 SP_APPABORT,并且接下来会绕开对 EndDoc 的调用。

打印假脱机

目前为止所讲的内容组成了打印处理的“前台”——由应用程序负责的部分。Windows 处理后台,是由 GDI、打印假脱机程序、打印机驱动程序以及其他 32 位打印子系统组件通力合作完成的。Windows 支持两类打印假脱机:EMF (增强型图元文件)打印假脱机和“原始”打印假脱机。如果 EMF 打印假脱机有效,通过打印机 DC 执行的 GDI 调用会写入硬盘的增强型图元文件,并保存在那里直到打印假脱机程序(在别的线程中运行)将命令调出并“交给”打印机驱动程序。如果选择了原始打印假脱机(PostScript 打印机的唯一选择),输出结果经过打印机驱动程序处理并以原始形式输出到磁盘上。打印假脱机也可以无效,这种情况下,当每次调用 EndPage 之后 GDI 命令会直接传送到打印机驱动程序。有了打印假脱机,应用程序不再等待打印机打印完每一页,从而缩短了返回应用程序需要的时间。通过将应用程序和打印机驱动程序的执行分开来,打印假脱机图元文件命令(不是原始打印机数据)进一步提高了返回应用程序的速度。

幸运的是应用程序可以毫无顾虑地忽略打印假脱机过程,把精力集中在前台。但是,在应用程序能真正做一些实际的打印工作之前,例如给输出结果编页码以及在 StartPage 和 EndPage 之间调用 GDI 并把每一页提交给打印机之前,还必须学习许多细节内容。有这些知识作背景,让我们看一下 MFC 能帮什么忙。

13.1.2 MFC 打印体系结构

MFC 简化的打印体系结构是促使 Windows 程序员从 SDK 转移到面向对象开发环境来的另一个原因。在给文档/视图应用程序添加打印功能时,可以忘掉前一节中的大部分程序代码。主框架将创建打印机打印设备描述表并在打印完成后删除它。主框架还调用 StartDoc 和 EndDoc 来开始和结束打印工作,还为每一页都用 StartPage 和 EndPage 将 GDI 调用括

起来。主框架甚至提供了显示打印工作状态和中止打印(在用户单击“取消”按钮后关闭打印操作)的对话框。在某些情况下,同一个 OnDraw 函数既可以将文档提交给屏幕,也可以提交给打印机以及打印预览窗口。

在文档/视图应用程序中实现打印功能的关键是一组 CView 虚函数,主框架要在打印处理的不同阶段调用它们。在表 13-2 中列出了这些函数。覆盖哪个以及在覆盖函数中做什么都取决于打印输出的内容。至少,总要覆盖 OnPreparePrinting 并调用 DoPreparePrinting,以便主框架可以显示“打印”对话框并创建打印机设备描述表。最简单的 OnPreparePrinting 覆盖函数如下:

```
BOOL CMyView::OnPreparePrinting(CPrintInfo* pInfo)
|
|   return DoPreparePrinting(pInfo);
|
```

如果 OnPreparePrinting 返回非零值,开始打印处理,返回 0 就取消打印工作。在下列情况下 DoPreparePrinting 返回 0: 用户通过在“打印”对话框中单击“取消”按钮取消了打印工作;没有安装打印机;或主框架不能创建打印机设备描述表。

表 13-2 主要 CView 打印覆盖函数

函数	说 明
OnPreparePrinting	在设置打印参数时调用。覆盖时可以用来调用 DoPreparePrinting 并给主框架提供打印页数(如果知道)和其他与打印有关的信息
OnBeginPrinting	在打印开始前调用。覆盖时可以用来分配字体和其他打印需要的资源
OnPrepareDC	在每一页被打印前调用。覆盖时可以用来指定视口原点位置并在 OnDraw 打印下页之前设置剪贴区域
OnPrint	在每一页被打印前调用。覆盖时可以用来打印页眉、页脚以及其他没有用 OnDraw 或不依赖 OnDraw 打印的页元素
OnEndPrinting	在打印结束后调用。覆盖时可以用来释放在 OnBeginPrinting 中分配的资源

在继续下面的学习之前,先介绍一下实现 MFC 应用程序打印功能的两种基本方法。第一种是让 OnDraw 既处理屏幕输出也处理打印输出。第二种是让 OnDraw 处理屏幕输出而让 OnPrint 处理打印输出。大多数有经验的 MFC 开发者都同意让 OnDraw 负全责的方法有些高估 OnDraw 的能力了。若用该方法几乎必然会要求您给 OnDraw 添加打印特有的程序段,通常最终您也会不得不在不同程度上覆盖 OnPrint 来打印页编号、页眉、页脚以及其他只有在打印纸上出现的页元素。因此,虽然视图的 OnDraw 函数可以在屏幕和打印机上输出结果,但通常更实际的是把打印机输出程序段放在 OnPrint 中,把屏幕输出程序段放在 OnDraw 中。在本章中将讨论这两种方法,但重点是后者。

详细讨论 OnPreparePrinting 函数

传递给 OnPreparePrinting 的 CPrintInfo 对象包含着描述打印工作参数的信息,其中包括最小和最大页号。最小和最大页编号默认值分别为 1 和 0xFFFF,0xFFFF 告诉主框架最大页编号还不知道。如果在调用 OnPreparePrinting 时应用程序知道文档包含多少页,就应该在调用 DoPreparePrinting 之前调用 CPrintInfo::SetMaxPage 来通知 MFC:

```
BOOL CMYView::OnPreparePrinting (CPrintInfo * pInfo)
{
    pInfo->SetMaxPage(10);
    return DoPreparePrinting(pInfo);
}
```

反过来,MFC 会在“打印”对话框的“To”框中显示最大页编号(本例中是 10)。

SetMinPage 和 SetMaxPage 是若干 CPrintInfo 成员函数中的两个,调用它们可以指定打印参数或向主框架查询用户输入的打印选项。GetFromPage 和 GetToPage 返回用户在“打印”对话框中输入的开始页和结束页的编号。由于是 DoPreparePrinting 显示对话框,所以要确保在 DoPreparePrinting 之后再调用它们。CPrintInfo 还包含几个公用数据成员,其中 m_pPD 变量指向已初始化的 CPrintDialog 对象,DoPreparePrinting 通过该对象来显示“打印”对话框。可以使用 m_pPD 指针在屏幕上显示对话框之前自定义“打印”对话框并通过调用 CPrintDialog 函数从对话框中获取信息或直接访问 CPrintDialog 数据成员。在本章稍后,有一个例子将说明如何做和为什么这样做。

OnBeginPrinting 和 OnEndPrinting 函数

通常最大页编号取决于从打印机输出的每一页可打印区域的大小。不幸的是,在用户选择打印机以及主框架创建打印机设备描述表之前,都无法确定实际的可打印区域。如果不在 OnPreparePrinting 中设置最大页编号,那么如果可能应该在 OnBeginPrinting 中设置。OnBeginPrinting 接收一个指向已初始化 CPrintInfo 结构的指针和一个指向 CDC 对象的指针,该对象代表调用 DoPreparePrinting 时主框架创建的打印机设备描述表。在 OnBeginPrinting 中可以调用 CDC::GetDeviceCaps 两次来确定可打印页区域的尺寸,一次用 HORZRES 参数来调用,一次用 VERTRES 参数来调用。下列 OnBeginPrinting 覆盖函数使用 GetDeviceCaps 以像素为单位来确定可打印页的区域并用此信息通知主框架文档中包含的页数:

```
void CMYView::OnBeginPrinting (CDC * pDC, CPrintInfo * pInfo)
{
    int nPageHeight = pDC->GetDeviceCaps (VERTRES);
    int nDocLength = GetDocument()->GetDocLength();
    int nMaxPage = max(1, (nDocLength + (nPageHeight-1)) /
```

```
nPageHeight);
pInfo->SetMaxPage(nMaxPage);
```

在此例中, `GetDocLength` 是文档类的成员函数, 返回以像素为单位的文档长度。 `CPrintInfo` 包含一个数据成员 `m_rectDraw`, 描述了逻辑坐标下可打印页区域, 但是注意不要在 `OnBeginPrinting` 中使用 `m_rectDraw`, 因为在 `OnBeginPrinting` 返回之前它还不会被初始化。

在 `OnPreparePrinting` 或 `OnBeginPrinting` 中调用 `SetMaxPage` 可以通知主框架应该调用多少次 `OnPrint` 来打印一页。如果在打印之前不可能(或不方便)确定文档的长度, 可以覆盖 `OnPrepareDC` 并在每次调用 `OnPrepareDC` 之前将 `CPrintInfo::m_bContinuePrinting` 设置为 `TRUE` 或 `FALSE` 来执行打印时分页。 `m_bContinuePrinting` 值为 `FALSE` 将结束打印工作。如果不调用 `SetMaxPage`, 主框架会假定文档只有一页长。所以, 如果不用 `SetMaxPage` 设置最大页编号, 就必须对多于一页的打印文档覆盖 `OnPrepareDC` 并设置 `m_bContinuePrinting` 值。

最好在 `OnBeginPrinting` 中创建字体和打印处理中用到的其他 GDI 资源。假设 `OnDraw` 用 GDI 字体在屏幕上输出文本而且字体高度基于当前屏幕度量单位。现在要将此字体的 WYSIWYG 版本输出到打印机上, 就必须创建一种独立的字体, 该字体与打印机度量单位而不是与屏幕度量单位成比例。通过在 `OnBeginPrinting` 中创建字体, 在 `OnEndPrinting` 中删除它, 就可以保证字体只在需要它的时候存在, 同时也避免了每次打印时重新创建它的开销。

`OnEndPrinting` 与 `OnBeginPrinting` 配对使用。最好在这里释放字体以及其他在 `OnBeginPrinting` 中分配的资源。如果没有需要释放的资源, 或如果在开始时没有覆盖 `OnBeginPrinting`, 那么可能也就没必要覆盖 `OnEndPrinting` 了。

OnPrepareDC 函数

对于打印文档的每一页, 都要调用一次 `OnPrepareDC`。要覆盖 `OnPrepareDC` 的一个原因就是执行上一节提到的打印分页功能。另一个原因是根据当前页编号计算新的视口原点以便 `OnDraw` 可以将当前页输出到打印机上。和 `OnBeginPrinting` 一样, `OnPrepareDC` 也接收指向设备描述表的指针和指向 `CPrintInfo` 对象的指针。与 `OnBeginPrinting` 不同的是, `OnPrepareDC` 是在屏幕重画之前在给打印机输出一页做准备的过程中被调用的。如果在屏幕重画之前调用了 `OnPrepareDC`, 引用屏幕 DC 的 CDC 指针和 `CPrintInfo` 指针将为 `NULL`。如果 `OnPrepareDC` 是作为打印过程的一部分被调用的, 引用打印机设备描述表的 CDC 指针和 `CPrintInfo` 指针则为非 `NULL` 值。在后一种情况下, 可以从 `CPrintInfo` 对象的公用 `m_nCurPage` 数据成员中获得待打印页的页编号。通过在参数列表中传递的 CDC 指针调用 `CDC::IsPrinting` 可以确定 `OnPrepareDC` 是针对屏幕还是针对打印机而被调用的。

下列 `OnPrepareDC` 函数在 y 方向上移动了视口原点, 以便使设备原点 (0,0)——被打印页左上角像素点——与文档当前页左上角的逻辑原点对应。 `m_nPageHeight` 是 `CMyView` 数据成员, 保存着可打印页的高度:


```

void CMyView::OnPrepareDC(CDC * pDC, CPrintInfo * pInfo)
{
    CView::OnPrepareDC(pDC, pInfo);
    if (pDC->IsPrinting()) { // If printing...
        int y = (pInfo->nCurPage-1) * m_nPageHeight;
        pDC->SetViewportOrg(0, -y);
    }
}

```

用这种方法设置视口原点确保了本来要绘制整个文档的 OnDraw 函数实际上只输出了与当前页相应的部分。本例的前提是您想使用整个可打印页区域。通常也有必要设置一个剪贴区域,用来使文档中被打印的部分小于页的整个可打印区域。用 CRgn::CreateRectRgn 创建矩形区域并用 CDC::SelectClipRgn 把它选入 DC 中作为剪贴矩形。

作为一种规则,您只有在使用 OnDraw 给屏幕和打印页绘制输出时才需要覆盖 OnPrepareDC。如果是使用 OnPrint 来完成打印任务就没必要覆盖 OnPrepareDC 了。在覆盖它时,应该在做其他事情之前先调用基类的 OnPrepareDC,以便有机会执行一些默认操作。如果视图类是从 CScrollView 派生的,那么调用基类的 OnPrepareDC 就变得非常重要,这是因为 CScrollView::OnPrepareDC 为屏幕 DC 设置了视口原点来与当前滚动位置匹配。在 CScrollView::OnPrepareDC 调用返回以后,DC 的映射模式被设置为在调用 SetScrollSizes 中指定的映射模式。如果您的视图类不是从 CScrollView 派生的,那么最好用 OnPrepareDC 调用 SetMapMode 来设置设备描述表的映射模式。

OnPrint 函数

在针对给定页调用了 OnPrepareDC 之后,主框架将调用 CView::OnPrint。和其他许多的 CView 打印函数一样,OnPrint 接收一个指向打印机设备描述表的指针和一个指向 CPrintInfo 对象的指针。在 Viewcore.cpp 的默认情况下,函数会检查 pDC 的有效性并调用 OnDraw:

```

void CView::OnPrint(CDC * pDC, CPrintInfo *)
{
    ASSERT_VALID(pDC);

    // Override and set printing variables based on page number
    OnDraw(pDC); // Call Draw
}

```

在覆盖 OnPrint 时要做什么工作(或者根本上覆盖它)完全取决于应用程序实现打印功能的方式。如果 OnDraw 既处理屏幕输出又处理打印输出,就要覆盖 OnPrint 来打印一些不会在屏幕上出现的页元素。下面给出的 OnPrint 函数调用了自己的成员函数 PrintHeader 在页的顶部打印页眉,调用另一个自己的成员函数 PrintPageNumber 在页的底部打印页编号,并调用

OnDraw 打印页的主体部分:

```
void CMYView::OnPrint (CDC* pDC, CPrintInfo* pInfo)
{
    Print-Header (pDC);
    PrintPageNumber (pDC, pInfo->m_nCurPage);
    // Set the viewport origin and/or clipping region before
    // calling OnDraw...
    OnDraw (pDC);
}
```

为了使 OnDraw 只绘制与当前页对应的文档部分,就要用 SetViewportOrg 或 SelectClipRgn 对打印机设备描述表做一些调整,而所有这些调整现在应该在 OnPrint 中而不是在 OnPrepareDC 中实现,这样可以避免影响页眉页编号。

如果选择了 OnPrint 来完成所有打印任务,就要覆盖 OnPrint 并在其中包含输出打印页的程序代码。要确定调用 OnPrint 来打印哪一页,可以查询 CPrintInfo::m_nCurPage。

CView::OnFilePrint 和其他命令处理程序

打印通常是从用户在“文件”菜单中选中了“打印”命令起开始的,因此 MFC 提供了 CView::OnFilePrint 函数,您可以通过视图中的消息映射表将它与 ID_FILE_PRINT 菜单项联系起来。图 13-1 说明了 OnFilePrint 被调用以及在打印处理过程中每一个 CView 打印虚函数被调用时所进行的操作。它还显示了与 Windows 打印体系结构交织在一起的 MFC 打印体系结构:如果将黑色矩形框(代表主框架所调用的 CView 虚函数)去掉,剩下的就是一个非常好的 Windows 打印模型示意图。注意如果程序在 Windows 95 或 Windows 98 中执行时,对于每一页都会调用 OnPrepareDC 两次。第一次调用 OnPrepareDC 是为了维持与 MFC 16 位版本的兼容性,因为在 16 位 Windows 中是 EndPage 而不是 StartPage 重新设置设备描述表,所以要在 StartPage 之前调用 OnPrepareDC。第二次调用 OnPrepareDC 是因为在 Windows 95 和 Windows 98 中,第一次调用 OnPrepareDC 对设备描述表所做的修改在调用 StartDoc 时会被取消。

MFC 还为“文件”菜单中的“打印预览”和“打印设置”命令提供了命令 ID 和默认的命令处理程序。“文件”菜单中的“打印预览”命令(ID = ID_FILE_PRINT_PREVIEW)由 CView::OnFilePrintPreview 处理,而“打印设置”命令(ID = ID_FILE_PRINT_SETUP)由 CWinApp::OnFilePrintSetup 处理。和 OnFilePrint 一样,这些命令处理程序并没有预先配置在所属类中的消息映射表中。要想使它们有效,必须亲自处理消息映射。如果使用 AppWizard 生成具有打印功能的应用程序框架,就不必亲自进行消息映射了。AppWizard 还将 ID_FILE_PRINT_DIRECT 映射给了 CView::OnFilePrint 从而使“直接”打印有效,所谓“直接”打印就是:用户不是从“文件”菜单中选择“打印”命令,而是从文档的上下文菜单选择“打印”命令或是将文档图标放置在打印机上而实现的一种打印操作。

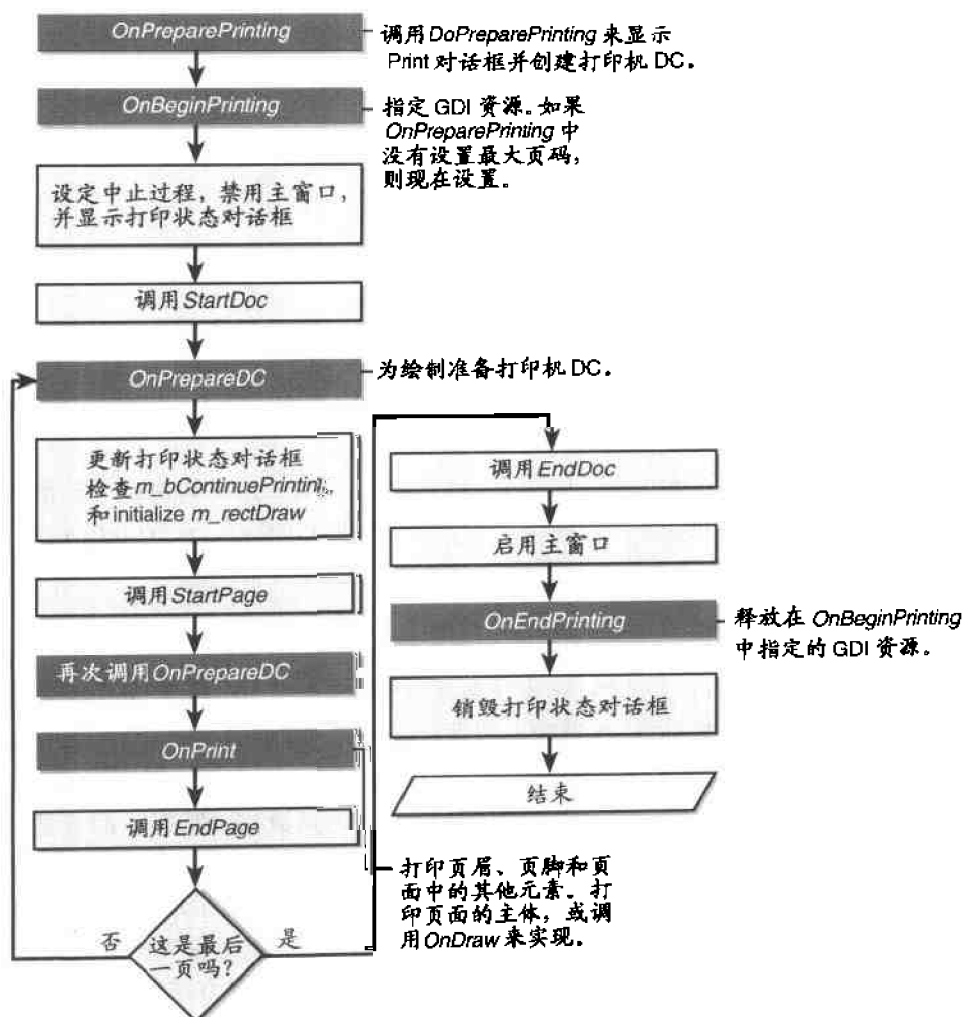


图 13-1 MFC 打印体系结构概述

13.1.3 打印预览

一旦给文档/视图应用程序赋予了打印功能, 再添加打印预览就简单了, 在“文件”菜单中添加“打印预览”命令 (ID = ID_FILE_PRINT_PREVIEW) 并在消息映射表中添加一项将此命令 ID 与 CView::OnFilePrintPreview 联系起来。许多程序代码用来支持 OnFilePrintPreview (参见 MFC 原程序代码文件 Viewprev.cpp), 但是在 OnFilePrintPreview 中的工作其实很简单。OnFilePrintPreview 取代了框架窗口并在其中添加了一个由特殊的 CScrollView 派生类 CPreviewView 创建的视图, 还增添了一个工具栏, 其中按钮有到下一页或到上一页、在一页和

两页视图间切换、视图缩放等等。CPreviewView::OnDraw 绘制一个代表被打印页的白色矩形(如果选择了两页视图则为两个矩形),设置一些尺寸比例参数以便白色矩形的可打印区域与实际打印纸的可打印区域匹配,并调用 OnPrint 在矩形中绘制输出。对应用程序而言,输出被发送给打印机了;在打印过程中被调用的函数同样会在打印预览中被调用。但实际上输出被发送给了打印预览窗口。

实现打印预览功能的奥秘在于:传递给 CView 打印函数的 pDC 参数所引用的设备描述表实际上是两个。每一个 CDC 对象都包含两个设备描述表句柄:一个针对“输出 DC”(m_hDC),另一个针对“属性 DC”(m_hAttribDC)。MFC 对于生成实际输出的调用使用输出 DC,而对于请求获取有关设备描述表信息(例如当前文本颜色、当前背景模式)的调用使用属性 DC。在大多数情况下,m_hDC 和 m_hAttribDC 保存着相同的设备描述表句柄。但是在打印预览过程中,m_hDC 引用可以预览打印页的屏幕 DC 而 m_hAttribDC 引用的是打印机 DC。结果如何?如果应用程序使用 GetDeviceCaps 或其他 CDC 函数向 GDI 查询打印机的功能或被打印页的属性,返回的信息确实是真实的,它来自打印机 DC。但是实际上所有的输出都给了屏幕 DC 了。

在用户关闭打印预览窗口时,主框架会调用 CView 虚函数 OnEndPrintPreview 来通知应用程序打印预览将要结束。默认 OnEndPrintPreview 操作将调用 OnEndPrinting,恢复最初视图,并关闭打印预览窗口。有时程序员会覆盖 OnEndPrintPreview,为的是将文档视图滚动到打印预览模式下所显示的最后--页(默认情况下,最初视图的滚动位置被保存起来了,以便在打印预览模式中的滚动不会影响到最初视图)。下列 OnEndPrintPreview 覆盖函数说明了对于 CScrollView 如何把最初视图中的滚动位置与打印预览窗口中的滚动位置联系起来的方法:

```
void CMYView::OnEndPrintPreview(CDC* pDC, CPrintInfo* pInfo,
    POINT point, CPreviewView* pView)
{
    UINT nPage = pInfo->m_nCurPage;
    POINT pt;
    // Convert nPage into a scroll position in pt.
    ScrollToPosition(pt);
    CScrollView::OnEndPrintPreview(pDC, pInfo, point, pView);
}
```

您必须亲自编写将当前页编号转换为滚动位置的程序代码。不要依靠传递给 OnEndPrintPreview 的 point 参数获取信息;在当今 MFC 版本中,point 总等于 (0,0)。应该在覆盖函数中调用基类的 OnEndPrintPreview 函数,以便主框架可以退出打印预览模式并恢复框架窗口的最初状态。

如果打印程序需要区分实际打印和在打印预览模式下的打印,可以检查 CPrintInfo 对象的 m_bPreview 数据成员,该对象在 OnBeginPrinting、OnPrint 以及其他打印可覆盖函数的调用

中被引用。如果文档在预览状态中, `m_bPreview` 为非零值, 否则为零。另外, 还可以通过检查 `CPrintInfo::m_nNumPreviewPages` 来确定显示的是一页还是两页。

13.2 只有打印功能的应用程序

图 13-2 所示的 EZPrint 应用程序说明了文档/视图应用程序支持打印和打印预览功能所必须的最少的工作量。

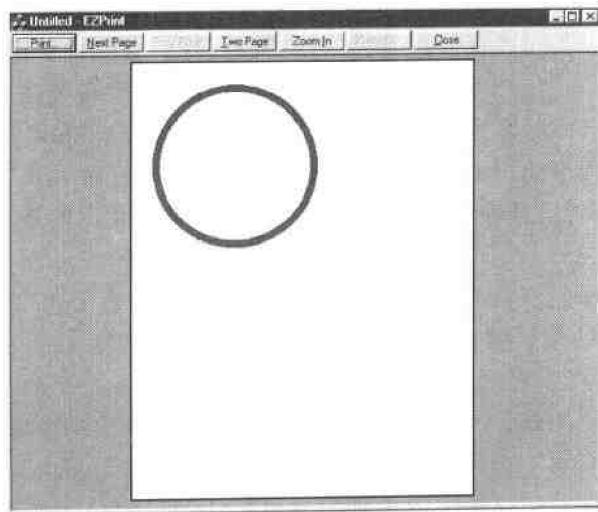


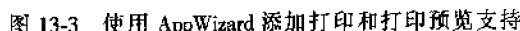
图 13-2 EZPrint 应用程序显示的打印预览

在 EZPrint “文档”中是一个外径 10 厘米(在 `MM_LOMETRIC` 映射模式下是 1 000 个单位)的蓝色圆环, 其中套着黄色的圆。应用程序的“文件”菜单中只有 4 个命令选项: “打印”、“打印预览”、“打印设置”以及“退出”。在 `CEZPrintView` 的消息映射表中, “打印”和“打印预览”命令映射到 `CView::OnFilePrint` 和 `CView::OnFilePrintPreview`, 而“打印设置”命令在 `CEZPrintApp` 的消息映射表中映射到 `CWinApp::OnFilePrintSetup`。AppWizard 完成了所有的消息映射。“打印”命令显示一个“打印”对话框, 其中用户可以指定打印选项, 如要求的打印机、打印范围和打印页数等。“打印预览”使应用程序进入打印预览模式。“打印设置”显示“打印设置”对话框。可以使用该对话框来选择打印机、选择打印纸大小以及打印页的方向——横向还是纵向。

我们使用 AppWizard 创建 EZPrint 项目。在 Step4 对话框(如图 13-3 所示)中选中 `Printing And Print Preview` 复选框来添加打印支持。选中此复选框可以提示 AppWizard 在它生成的程序代码中要进行 3 个调整工作:

- 在“文件”菜单中添加“打印”、“打印预览”和“打印设置”命令。

- AppWizard 的 OnPreparePrinting 函数包含对 DoPreparePrinting 的调用。它的 OnBeginPrinting 和 OnEndPrinting 函数什么也不做,因此如果不使用它们就可以将其删除。我们在程序中保留了它们,但 EZPrint 没有它们也一样正常工作。所有 EZPrint 的打印程序代码都可以在视图类中找到,图 13-4 给出了程序源代码。



EZPrintView.h

[illegible]

```

# if !defined(
    AFX_EZPRINTVIEW_H_ 3A83FD5D_A3E6_11D2_8E53_006008A82731__INCLUDED_)
# define AFX_EZPRINTVIEW_H_ 3A83FD5D_A3E6_11D2_8E53_006008A82731__INCLUDED_

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

class CEZPrintView : public CView
{
protected: // create from serialization only
    CEZPrintView();
    DECLARE_DYNCREATE(CEZPrintView)

// Attributes
public:
    CEZPrintDoc * GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CEZPrintView)
public:
    virtual void OnDraw(CDC * pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo * pInfo);
    virtual void OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo);
    virtual void OnEndPrinting(CDC * pDC, CPrintInfo * pInfo);
    //}}AFX_VIRTUAL

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

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CEZPrintView)
    // 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 EZPrintView.cpp
    inline CEZPrintDoc* CEZPrintView::GetDocument()
    { return (CEZPrintDoc*)m_pDocument; }
    #endif

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

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

    #endif
    // !defined(
    //     AFX_EZPRINTVIEW_H_ 3A83F0ED_A3E6_11D2_8E53_006008A82731__INCLUDED_)

```

EZPrintView.cpp

```

// EZPrintView.cpp : implementation of the CEZPrintView class
//
#include "stdafx.h"
#include "EZPrint.h"

#include "EZPrintDoc.h"
#include "EZPrintview.h"

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

////////////////////////////////////
// CEZPrintView

IMPLEMENT_DYNCREATE(CEZPrintView, CView)

BEGIN_MESSAGE_MAP(CEZPrintView, CView)
    //||AFX_MSG_MAP(CEZPrintView)
    // 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
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

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

```


[illegible]

```

#ifdef DEBUG
void CEZPrintView::AssertValid() const
{
    CView::AssertValid();
}

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

CEZPrintDoc* CEZPrintView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument -> IsKindOf(RUNTIME_CLASS(CEZPrintDoc)));
    return (CEZPrintDoc*)m_pDocument;
}

#endif // DEBUG

////////////////////////////////////
// CEZPrintView message handlers

```

图 13-4 EZPrint 应用程序

13.2.1 黑白打印预览

MFC 的打印预览支持并不完美。即使与 PC 连接的打印机处于黑白模式下, EZPrint 的打印预览页也会显示全真彩色的圆(当然如果在彩色打印机上打印就会打印出有颜色的圆)。如果 OnPrint 或 OnDraw 被调用时下列两个条件成立, 就可以改动打印预览程序代码, 进而实现灰度渲染:

- pInfo -> m_bPreview 为非零值 (OnPrint) 或
pDC -> m_hDC 不等于 pDC -> m_hAttribDC (OnDraw)
- pDC -> GetDeviceCaps (NUMCOLORS) 返回 2, 说明是单色打印机。

用下列公式可以将 RGB 颜色转换为灰色调:

$$r/g/b = (\text{red} * 0.30) + (\text{green} * 0.59) + (\text{blue} * 0.11)$$

下列语句创建了一个灰色画刷, 它可以在屏幕上模拟黄色 (RGB (255, 255, 0)) 在单色输出设备上的样子:

```
CBrush brush (RGB (227, 227, 227));
```

值 227 是将颜色成分值 255、255 和 0 代入颜色转换公式后计算得到的。

如果要得到黑白打印预览的简单示例, 可以将 EZPrint 的 CPrintView::OnDraw 函数中的

下列语句

```
CPen pen (PS_SOLID, 50, RGB (0, 0, 255));
CBrush brush (RGB (255, 255, 0));
```

用

```
BOOL bMono = (pDC->GetDeviceCaps (NUMCOLORS) == 2) &&
    (pDC->m_hDC != pDC->m_hAttribDC); // True only for preview mode.
CPen pen (PS_SOLID, 50, bMono ? RGB (28, 28, 28) : RGB (0, 0, 255));
CBrush brush (bMono ? RGB (227, 227, 227) : RGB (255, 255, 0));
```

替换。如果默认打印机为黑白模式,打印预览就会用灰度渲染。CPrintInfo 信息无效时,通过比较 m_hDC 和 m_hAttribDC 就可以巧妙查知打印预览的模式。

13.3 复杂的打印应用程序

EZPrint 用作起步教学还可以,实际上在现实世界中没有这样的应用程序。因为文档中只包含一页,所以它没必要去处理分页工作。在每次调用 OnDraw 时都创建需要的 GDI 资源,所以也不需要 OnBeginPrinting 和 OnEndPrinting 来分配打印机指定的资源。由于在 EZPrint 中不用区分打印视图和屏幕视图,所以根本不需要去覆盖 OnPrepareDC 和 OnPrint。

图 13-5 所示的 HexDump 应用程序可能会更好地代表您要编写的应用程序。HexDump 是一个十六进制查看程序,它可以以二进制形式显示任何文件的内容。被打印页在顶部有一个包含文件名(如果有地方则以路径名开头)的页眉和一个页编号。页眉下有绍的水平下划线。这条线是用 CDC::MoveTo 和 CDC::LineTo 画的;所有其他输出用 CDC::TextOut 实现。图 13-6 显示了打印预览模式下的一页文档。在打印文档时,HexDump 会查询打印机来获取可打印页的尺寸并根据它相应调整输出。页高度被用来计算每一页打印的行数,不管页尺寸和输出方向怎样,页宽度都用来使输出在水平方向上居中。

CHexView::OnDraw 生成所有 HexDump 的屏幕输出。要重绘视图,OnDraw 调用 CDC::GetClipBox 来标识需要重绘的矩形,将矩形顶部和底部的 y 坐标转换为开始和结束的行号,并只绘制需要重绘的行。在输出中使用的字体是在 CHexView::OnCreate 中初始化了的 10 点阵 Courier New 屏幕字体。因为 CHexView 是从 CScrollView 派生来的,所以当前滚动位置已经在输出中自动考虑进去了。由于 OnDraw 做了最少的打印工作,所以即使文档非常大,滚动操作还是可接受的。如果没有优化 OnDraw,在加载了大型文档以后 CScrollView 操作就会变得很迟钝,如果想实验一下,可以重新编写 OnDraw 使它在每次被调用时都绘制整个文档。需要做的工作只是用

```
UINT nStart = 0;
UINT nEnd = m_nLinesTotal - 1;
```

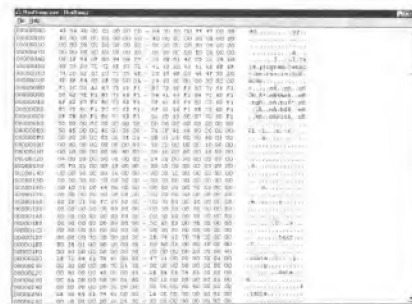


图 13-5 HexDump 显示文件的二进制视图



图 13-6 HexDump 的打印预览