#include "stdafx.h"

#include "MainFrm.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] =\_\_FILE\_\_;

#endif

// CMainFrame

IMPLEMENT\_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN\_MESSAGE\_MAP(CMainFrame, CMDIFrameWnd)

//{{AFX\_MSG\_MAP(CMainFrame)

// NOTE - the ClassWizard will add and remove mapping macros here.

// DO NOT EDIT what you see in these blocks of generated code !

ON\_WM\_CREATE()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

static UINT indicators[] =

{

ID\_SEPARATOR, // status line indicator

ID\_INDICATOR\_CAPS,

ID\_INDICATOR\_NUM,

ID\_INDICATOR\_SCRL,

};

// CMainFrame construction/destruction

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)

return -1;

if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar.LoadToolBar(IDR\_MAINFRAME))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndStatusBar.Create(this) ||

!m\_wndStatusBar.SetIndicators(indicators,

sizeof(indicators)/sizeof(UINT)))

{

TRACE0("Failed to create status bar\n");

return -1; // fail to create

}

// TODO: Delete these three lines if you don't want the toolbar to

// be dockable

m\_wndToolBar.EnableDocking(CBRS\_ALIGN\_ANY);

EnableDocking(CBRS\_ALIGN\_ANY);

DockControlBar(&m\_wndToolBar);

return 0;

}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

{

if( !CMDIFrameWnd::PreCreateWindow(cs) )

return FALSE;

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

return TRUE;

}

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

// CMainFrame diagnostics

#ifdef \_DEBUG

void CMainFrame::AssertValid() const

{

CMDIFrameWnd::AssertValid();

}

void CMainFrame::Dump(CDumpContext& dc) const

{

CMDIFrameWnd::Dump(dc);

}

#endif //\_DEBUG

// test1.cpp : Defines the class behaviors for the application.

//

#include "stdafx.h"

#include "test1.h"

#include "MainFrm.h"

#include "ChildFrm.h"

#include "test1Doc.h"

#include "test1View.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

BEGIN\_MESSAGE\_MAP(CTest1App, CWinApp)

//{{AFX\_MSG\_MAP(CTest1App)

ON\_COMMAND(ID\_APP\_ABOUT, OnAppAbout)

// 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 file based document commands

ON\_COMMAND(ID\_FILE\_NEW, CWinApp::OnFileNew)

ON\_COMMAND(ID\_FILE\_OPEN, CWinApp::OnFileOpen)

// Standard print setup command

ON\_COMMAND(ID\_FILE\_PRINT\_SETUP, CWinApp::OnFilePrintSetup)

END\_MESSAGE\_MAP()

// CTest1App construction

CTest1App::CTest1App()

{

// TODO: add construction code here,

// Place all significant initialization in InitInstance

}

// The one and only CTest1App object

CTest1App theApp;

// CTest1App initialization

BOOL CTest1App::InitInstance()

{

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

// Change the registry key under which our settings are stored.

// TODO: You should modify this string to be something appropriate

// such as the name of your company or organization.

SetRegistryKey(\_T("Local AppWizard-Generated Applications"));

LoadStdProfileSettings(); // Load standard INI file options (including MRU)

// Register the application's document templates. Document templates

// serve as the connection between documents, frame windows and views.

CMultiDocTemplate\* pDocTemplate;

pDocTemplate = new CMultiDocTemplate(

IDR\_TEST1TYPE,

RUNTIME\_CLASS(CTest1Doc),

RUNTIME\_CLASS(CChildFrame), // custom MDI child frame

RUNTIME\_CLASS(CTest1View));

AddDocTemplate(pDocTemplate);

// create main MDI Frame window

CMainFrame\* pMainFrame = new CMainFrame;

if (!pMainFrame->LoadFrame(IDR\_MAINFRAME))

return FALSE;

m\_pMainWnd = pMainFrame;

// Parse command line for standard shell commands, DDE, file open

CCommandLineInfo cmdInfo;

ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line

if (!ProcessShellCommand(cmdInfo))

return FALSE;

// The main window has been initialized, so show and update it.

pMainFrame->ShowWindow(m\_nCmdShow);

pMainFrame->UpdateWindow();

return TRUE;

}

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog

{

public:

CAboutDlg();

// Dialog Data

//{{AFX\_DATA(CAboutDlg)

enum { IDD = IDD\_ABOUTBOX };

//}}AFX\_DATA

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CAboutDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

// Implementation

protected:

//{{AFX\_MSG(CAboutDlg)

// No message handlers

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

{

//{{AFX\_DATA\_INIT(CAboutDlg)

//}}AFX\_DATA\_INIT

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CAboutDlg)

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialog)

//{{AFX\_MSG\_MAP(CAboutDlg)

// No message handlers

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

// App command to run the dialog

void CTest1App::OnAppAbout()

{

CAboutDlg aboutDlg;

aboutDlg.DoModal();

}

// CTest1App message handlers

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)

return -1;

if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar.LoadToolBar(IDR\_MAINFRAME))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndStatusBar.Create(this) ||

!m\_wndStatusBar.SetIndicators(indicators,

sizeof(indicators)/sizeof(UINT)))

{

TRACE0("Failed to create status bar\n");

return -1; // fail to create

}

// TODO: Delete these three lines if you don't want the toolbar to

// be dockable

m\_wndToolBar.EnableDocking(CBRS\_ALIGN\_ANY);

EnableDocking(CBRS\_ALIGN\_ANY);

DockControlBar(&m\_wndToolBar);

return 0;

}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

{

if( !CMDIFrameWnd::PreCreateWindow(cs) )

return FALSE;

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

return TRUE;

}

#ifdef \_DEBUG

void CMainFrame::AssertValid() const

{

CMDIFrameWnd::AssertValid();

}

void CMainFrame::Dump(CDumpContext& dc) const

{

CMDIFrameWnd::Dump(dc);

}

#endif //\_DEBUG

void CMainFrame::OnRecordStart()

{

// TODO: Add your command handler code here

MessageBeep((UINT)(-1));

}

void CMainFrame::OnRecordStop()

{

// TODO: Add your command handler code here

MessageBeep((UINT)(-1));

}

void CMainFrame::OnUpdateRecordStart(CCmdUI\* pCmdUI)

{

// TODO: Add your command update UI handler code here

pCmdUI->Enable(!m\_bWorking);

}

void CMainFrame::OnUpdateRecordStop(CCmdUI\* pCmdUI)

{

// TODO: Add your command update UI handler code here

pCmdUI->Enable(m\_bWorking) ;

}

void CMainFrame::OnHighQuality()

{

// TODO: Add your command handler code here

m\_bHighQuality=TRUE;

}

void CMainFrame::OnLowQuality()

{

// TODO: Add your command handler code here

m\_bHighQuality=FALSE;

}

void CMainFrame::OnUpdateHighQuality(CCmdUI\* pCmdUI)

{

// TODO: Add your command update UI handler code here

pCmdUI->SetCheck(m\_bHighQuality);

}

void CMainFrame::OnUpdateLowQuality(CCmdUI\* pCmdUI)

{

// TODO: Add your command update UI handler code here

pCmdUI->SetCheck(!m\_bHighQuality);

}

void CMainFrame::OnViewToolbar()

{

// TODO: Add your command handler code here

m\_wndToolBar.ShowWindow(m\_wndToolBar.IsWindowVisible()?

SW\_HIDE:SW\_SHOW);

RecalcLayout();

}

void CMainFrame::OnUpdateViewToolbar(CCmdUI\* pCmdUI)

{

// TODO: Add your command update UI handler code here

pCmdUI->SetCheck(m\_wndToolBar.IsWindowVisible());

}

IMPLEMENT\_DYNCREATE(CMyRecordView, CView)

BEGIN\_MESSAGE\_MAP(CMyRecordView, CView)

//{{AFX\_MSG\_MAP(CMyRecordView)

// 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()

CMyRecordView::CMyRecordView()

{

// TODO: add construction code here

}

CMyRecordView::~CMyRecordView()

{

}

BOOL CMyRecordView::PreCreateWindow(CREATESTRUCT& cs)

{

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

return CView::PreCreateWindow(cs);

}

void CMyRecordView::OnDraw(CDC\* pDC)

{

CRecordDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

// TODO: add draw code for native data here

}

BOOL CMyRecordView::OnPreparePrinting(CPrintInfo\* pInfo)

{

// default preparation

return DoPreparePrinting(pInfo);

}

void CMyRecordView::OnBeginPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: add extra initialization before printing

}

void CMyRecordView::OnEndPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: add cleanup after printing

}

#ifdef \_DEBUG

void CMyRecordView::AssertValid() const

{

CView::AssertValid();

}

void CMyRecordView::Dump(CDumpContext& dc) const

{

CView::Dump(dc);

}

CRecordDoc\* CMyRecordView::GetDocument() // non-debug version is inline

{

ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CRecordDoc)));

return (CRecordDoc\*)m\_pDocument;

}

#endif //\_DEBUG

IMPLEMENT\_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN\_MESSAGE\_MAP(CMainFrame, CFrameWnd)

//{{AFX\_MSG\_MAP(CMainFrame)

// NOTE - the ClassWizard will add and remove mapping macros here.

// DO NOT EDIT what you see in these blocks of generated code !

ON\_WM\_CREATE()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

static UINT indicators[] =

{

ID\_SEPARATOR, // status line indicator

ID\_INDICATOR\_CAPS,

ID\_INDICATOR\_NUM,

ID\_INDICATOR\_SCRL,

};

CMainFrame::CMainFrame()

{

// TODO: add member initialization code here

CMainFrame::~CMainFrame()

{

}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CFrameWnd::OnCreate(lpCreateStruct) == -1)

return -1;

if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar.LoadToolBar(IDR\_MAINFRAME))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndStatusBar.Create(this) ||

!m\_wndStatusBar.SetIndicators(indicators,

sizeof(indicators)/sizeof(UINT)))

{

TRACE0("Failed to create status bar\n");

return -1; // fail to create

}

// TODO: Delete these three lines if you don't want the toolbar to

// be dockable

m\_wndToolBar.EnableDocking(CBRS\_ALIGN\_ANY);

EnableDocking(CBRS\_ALIGN\_ANY);

DockControlBar(&m\_wndToolBar);

return 0;

}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

{

if( !CFrameWnd::PreCreateWindow(cs) )

return FALSE;

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

return TRUE;

}

#ifdef \_DEBUG

void CMainFrame::AssertValid() const

{

CFrameWnd::AssertValid();

}

void CMainFrame::Dump(CDumpContext& dc) const

{

CFrameWnd::Dump(dc);

}

#endif //\_DEBUG

int main(int argc, char \*\*argv)

{

Mat gau, gau1;

int start = GetTickCount();

GaussianBlur(src, gau, Size(0,0), 16);

cout << "GaussianBlur: "<< GetTickCount()-start << "ms"<<endl;

imshow("gau", gau);

// cvSmooth(gray, gau1, 2, 0,0, 0.0, 0.0);

// GaussianBlur(gray, gau1, Size(0,0), 0.0);

// imshow("gau1", gau1);

Mat gauss;

start = GetTickCount();

GaussianTemplateSmooth(gray, gauss, 0.0);

cout << "GaussianTemplateSmooth: "<< GetTickCount()-start << "ms"<<endl;

imshow("gauss", gauss);

// imwrite("dst.jpg", dst);

// imwrite("gauss.jpg", gauss);

//0.84089642

Mat gs;

start = GetTickCount();

GaussianSmooth(src, gs, 16);

cout << "GaussianSmooth: "<< GetTickCount()-start << "ms"<<endl;

imshow("gs", gs);

// imwrite("small.jpg", small);

// imwrite("gs.jpg", gs);

Mat g1;

start = GetTickCount();

GaussianSmooth2D(gray, g1, 16);

cout << "GaussianSmooth2D: "<< GetTickCount()-start << "ms"<<endl;

imshow("g1", g1);

// imwrite("g1.jpg", g1);

if(src.empty() || gray.empty() || dst.empty())

return -1;

// imshow("src", src);

imshow("gray", gray);

imshow("dst", dst);

waitKey();

return 0;

}

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

// CTestContainer98View

IMPLEMENT\_DYNCREATE( CTestContainer98View, CView )

BEGIN\_MESSAGE\_MAP( CTestContainer98View, CView )

//{{AFX\_MSG\_MAP(CTestContainer98View)

ON\_WM\_DESTROY()

ON\_WM\_SETFOCUS()

ON\_WM\_SIZE()

ON\_COMMAND(ID\_OLE\_INSERT\_NEW, OnInsertObject)

ON\_COMMAND(ID\_CANCEL\_EDIT\_CNTR, OnCancelEditCntr)

ON\_WM\_LBUTTONDOWN()

ON\_WM\_LBUTTONDBLCLK()

ON\_WM\_SETCURSOR()

ON\_COMMAND(ID\_EDIT\_CLEAR, OnEditClear)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_CLEAR, OnUpdateEditClear)

ON\_WM\_RBUTTONDOWN()

ON\_COMMAND(ID\_OPTIONS\_DESIGNMODE, OnOptionsDesignMode)

ON\_UPDATE\_COMMAND\_UI(ID\_OPTIONS\_DESIGNMODE, OnUpdateOptionsDesignMode)

ON\_WM\_CREATE()

ON\_WM\_LBUTTONUP()

ON\_WM\_MOUSEMOVE()

ON\_WM\_RBUTTONDBLCLK()

ON\_WM\_RBUTTONUP()

ON\_WM\_CAPTURECHANGED()

ON\_WM\_KILLFOCUS()

ON\_WM\_KEYDOWN()

ON\_WM\_KEYUP()

ON\_WM\_CHAR()

ON\_WM\_CANCELMODE()

ON\_COMMAND(ID\_CONTROL\_SAVETOSTREAM, OnControlSaveToStream)

ON\_COMMAND(ID\_EDIT\_INSERTFROMSTREAM, OnEditInsertFromStream)

ON\_COMMAND(ID\_CONTROL\_SAVETOPROPERTYBAG, OnControlSaveToPropertyBag)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_SAVETOPROPERTYBAG, OnUpdateControlSaveToPropertyBag)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_SAVETOSTREAM, OnUpdateControlSaveToStream)

ON\_COMMAND(ID\_CONTROL\_INVOKEMETHODS, OnControlInvokeMethods)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_INVOKEMETHODS, OnUpdateControlInvokeMethods)

ON\_COMMAND(ID\_CONTROL\_LOGGING, OnControlLogging)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_LOGGING, OnUpdateControlLogging)

ON\_COMMAND(ID\_CONTROL\_ACTIVATE, OnControlActivate)

ON\_COMMAND(ID\_CONTROL\_DEACTIVATE, OnControlDeactivate)

ON\_COMMAND(ID\_CONTROL\_UIACTIVATE, OnControlUIActivate)

ON\_COMMAND(ID\_CONTROL\_UIDEACTIVATE, OnControlUIDeactivate)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_UIDEACTIVATE, OnUpdateControlUIDeactivate)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_UIACTIVATE, OnUpdateControlUIActivate)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_ACTIVATE, OnUpdateControlActivate)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_DEACTIVATE, OnUpdateControlDeactivate)

ON\_COMMAND(ID\_EDIT\_BRINGTOFRONT, OnEditBringToFront)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_BRINGTOFRONT, OnUpdateEditBringToFront)

ON\_COMMAND(ID\_EDIT\_SENDTOBACK, OnEditSendToBack)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_SENDTOBACK, OnUpdateEditSendToBack)

ON\_WM\_CONTEXTMENU()

ON\_COMMAND(ID\_EDIT\_PROPERTIES, OnEditProperties)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_PROPERTIES, OnUpdateEditProperties)

ON\_COMMAND(ID\_EDIT\_SELECTALL, OnEditSelectAll)

ON\_COMMAND(ID\_EDIT\_COPY, OnEditCopy)

ON\_COMMAND(ID\_EDIT\_PASTE, OnEditPaste)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_COPY, OnUpdateEditCopy)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_PASTE, OnUpdateEditPaste)

ON\_COMMAND(ID\_EDIT\_CUT, OnEditCut)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_CUT, OnUpdateEditCut)

ON\_COMMAND(ID\_CONTROL\_SAVETOSTORAGE, OnControlSaveToStorage)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_SAVETOSTORAGE, OnUpdateControlSaveToStorage)

ON\_COMMAND(ID\_EDIT\_INSERTCONTROLFROMSTORAGE, OnEditInsertControlFromStorage)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_SELECTALL, OnUpdateEditSelectAll)

ON\_COMMAND(ID\_CONTROL\_DRAWMETAFILE, OnControlDrawMetaFile)

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL\_DRAWMETAFILE, OnUpdateControlDrawMetaFile)

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

CLIPFORMAT CTestContainer98View::s\_cfObjectDescriptor = NULL;

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

// CTestContainer98View construction/destruction

CTestContainer98View::CTestContainer98View() :

m\_pSelection( NULL ),

// m\_tDoingDrag( FALSE ),

m\_pItemWithCapture( NULL ),

m\_pItemWithFocus( NULL ),

m\_pItemToDeactivateOnLeave( NULL ),

m\_pInactiveDropTarget( NULL ),

m\_pWindowlessDropTarget( NULL ),

m\_dwLastDropKeyState( 0 )

{

if( s\_cfObjectDescriptor == NULL )

{

s\_cfObjectDescriptor = CLIPFORMAT( ::RegisterClipboardFormat( \_T(

"Object Descriptor" ) ) );

}

}

CTestContainer98View::~CTestContainer98View()

{

}

BOOL CTestContainer98View::PreCreateWindow( CREATESTRUCT& cs )

{

WNDCLASS wc;

if( !CView::PreCreateWindow( cs ) )

{

return( FALSE );

}

GetClassInfo( AfxGetInstanceHandle(), cs.lpszClass, &wc );

cs.lpszClass = AfxRegisterWndClass( 0, wc.hCursor, wc.hbrBackground,

wc.hIcon );

cs.style |= WS\_CLIPCHILDREN;

return( TRUE );

}

void CTestContainer98View::ClearSelection()

{

m\_pSelection = NULL;

m\_lpSelectedItems.RemoveAll();

}

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

// CTestContainer98View drawing

void CTestContainer98View::OnDraw( CDC\* pDC )

{

POSITION posItem;

CTestContainer98Doc\* pDoc;

CTestContainer98Item\* pItem;

pDoc = GetDocument();

ASSERT\_VALID( pDoc );

if( pDoc->UseTwoPassDrawing() )

{

// Draw the opaque parts of the controls from front to back.

TCTrace( TRACELEVEL\_VERBOSE, L"Starting drawing pass 1.\n" );

posItem = pDoc->GetStartPosition();

while( posItem != NULL )

{

pItem = (CTestContainer98Item\*)pDoc->GetNextItem( posItem );

// Push the state of the DC onto the stack. This is to save the

// current clip region for when we do the second pass.

pDC->SaveDC();

// The item will exclude any opaque areas from the clip region of the

// DC.

pItem->DrawFrontToBack( pDC, pItem->m\_rect );

if( pDoc->UseSlowDrawing() )

{

Sleep( 1000 );

}

}

// Draw the rest of each control from back to front, clipping out opaque

// regions of any controls in front.

TCTrace( TRACELEVEL\_VERBOSE, L"Starting drawing pass 2.\n" );

posItem = pDoc->GetTailPosition();

while( posItem != NULL )

{

pItem = (CTestContainer98Item\*)pDoc->GetPrevItem( posItem );

// Draw the tracker for the item.

CRectTracker tracker;

SetupTracker( pItem, &tracker );

tracker.Draw( pDC );

// Pop the state of the DC off the stack. This makes sure the

// clipping region excludes only the opaque regions of all the

// controls in front of this one.

pDC->RestoreDC( -1 );

pItem->DrawBackToFront( pDC, pItem->m\_rect );

if( pDoc->UseSlowDrawing() )

{

Sleep( 1000 );

}

}

}

else

{

// Draw the OLE items from the list.

posItem = pDoc->GetTailPosition();

while( posItem != NULL )

{

pItem = (CTestContainer98Item\*)pDoc->GetPrevItem( posItem );

// Draw the tracker for the item.

CRectTracker tracker;

SetupTracker( pItem, &tracker );

tracker.Draw( pDC );

// Draw the item.

pItem->Draw( pDC, pItem->m\_rect );

if( pDoc->UseSlowDrawing() )

{

Sleep( 1000 );

}

}

}

}

void CTestContainer98View::OnInitialUpdate()

{

CView::OnInitialUpdate();

m\_pSelection = NULL;

m\_lpSelectedItems.RemoveAll();

CTestContainer98App \* pApp = (CTestContainer98App \* )AfxGetApp();

ENSURE(pApp);

if(pApp->m\_options.m\_strDSMFile != \_T(""))

{

CTestContainer98Doc \* pDoc = GetDocument();

ENSURE(pDoc);

pDoc->RunAutomationScript(pApp->m\_options.m\_strDSMFile);

}

}

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

// CTestContainer98View printing

void CTestContainer98View::OnDestroy()

{

// Deactivate the item on destruction; this is important

// when a splitter view is being used.

CView::OnDestroy();

COleClientItem\* pActiveItem = GetDocument()->GetInPlaceActiveItem( this );

if( (pActiveItem != NULL) && (pActiveItem->GetActiveView() == this) )

{

pActiveItem->Deactivate();

ASSERT( GetDocument()->GetInPlaceActiveItem( this ) == NULL );

}

}

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

// OLE Client support and commands

BOOL CTestContainer98View::IsSelected( const CObject\* pDocItem ) const

{

// The implementation below is adequate if your selection consists of

// only CTestContainer98Item objects. To handle different selection

// mechanisms, the implementation here should be replaced.

// TODO: implement this function that tests for a selected OLE client item

return( pDocItem == m\_pSelection );

}

void CTestContainer98View::OnInsertObject()

{

CTestContainer98Item\* pItem;

CTestContainer98Doc\* pDoc;

CInsertControlDlg dlg;

int nResult; nResult = (int)dlg.DoModal();

if( nResult != IDOK )

{

return;

}

BeginWaitCursor();

pItem = NULL;

try

{

// Create new item connected to this document.

pDoc = GetDocument();

pItem = pDoc->AddControl( dlg.m\_clsid, NULL );

ASSERT\_VALID( pItem );

// As an arbitrary user interface design, this sets the selection to the

// last item inserted.

SetSelection( pItem );

pItem->InvalidateItem();

}

catch( COleException\* pException )

{

CString strErrorMessage;

\_com\_error error( pException->m\_sc );

pException->Delete();

if( pItem != NULL )

{

ASSERT\_VALID( pItem );

pItem->Delete();

}

AfxFormatString1( strErrorMessage, IDS\_CREATEFAILED,

error.ErrorMessage() );

AfxMessageBox( strErrorMessage );

}

catch( CException\* pException )

{

pException->Delete();

if( pItem != NULL )

{

ASSERT\_VALID( pItem );

pItem->Delete();

}

AfxMessageBox( IDP\_FAILED\_TO\_CREATE );

}

EndWaitCursor();

}

// The following command handler provides the standard keyboard

// user interface to cancel an in-place editing session. Here,

// the container (not the server) causes the deactivation.

void CTestContainer98View::OnCancelEditCntr()

{

// Close any in-place active item on this view.

COleClientItem\* pActiveItem = GetDocument()->GetInPlaceActiveItem( this );

if( pActiveItem != NULL )

{

pActiveItem->Close();

}

ASSERT( GetDocument()->GetInPlaceActiveItem( this ) == NULL );

}

// Special handling of OnSetFocus and OnSize are required for a container

// when an object is being edited in-place.

void CTestContainer98View::OnSetFocus( CWnd\* pOldWnd )

{

COleClientItem\* pActiveItem = GetDocument()->GetInPlaceActiveItem( this );

if( (pActiveItem != NULL) && (pActiveItem->GetItemState() ==

COleClientItem::activeUIState) )

{

// need to set focus to this item if it is in the same view

CWnd\* pWnd = pActiveItem->GetInPlaceWindow();

if( pWnd != NULL )

{

pWnd->SetFocus(); // don't call the base class

return;

}

}

CView::OnSetFocus( pOldWnd );

}

void CTestContainer98View::OnSize( UINT nType, int cx, int cy )

{

COleClientItem\* pActiveItem;

CView::OnSize( nType, cx, cy );

pActiveItem = GetDocument()->GetInPlaceActiveItem( this );

if( pActiveItem != NULL )

{

pActiveItem->SetItemRects();

}

}

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

// CTestContainer98View diagnostics

#ifdef \_DEBUG

void CTestContainer98View::AssertValid() const

{

CView::AssertValid();

}void CTestContainer98View::Dump( CDumpContext& dc ) const

{

CView::Dump( dc );

}

// non-debug version is inline.

CTestContainer98Doc\* CTestContainer98View::GetDocument()

{

ASSERT( m\_pDocument->IsKindOf( RUNTIME\_CLASS( CTestContainer98Doc ) ) );

return( (CTestContainer98Doc\*)m\_pDocument );

}

#endif //\_DEBUG

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

// CTestContainer98View message handlers

// Method: HitTestItems

// Description: Determines the topmost item that contains the given point.

// Parameters:

// point - (IN) The point to hit test.

// pdwHitType - (OUT) Which part of the item was hit. Possible values are:

// HITTYPE\_NONE - No item was hit.

// HITTYPE\_OBJECT - The item itself was hit.

// HITTYPE\_BORDER - The grab border of the item was hit.

// Return Value: Topmost item containing the point, or NULL if the point is not

// inside any of the items.

CTestContainer98Item\* CTestContainer98View::HitTestItems( CPoint point,

DWORD\* pdwHitType )

{

CTestContainer98Doc\* pDoc;

CTestContainer98Item\* pItem;

POSITION posItem;

CRect rectBorder;

DWORD dwHitResult;

pDoc = GetDocument();

\*pdwHitType = HITTYPE\_NONE; // Assume that we don't hit anything.

// Traverse the list of controls from front to back.

posItem = pDoc->GetStartPosition();

while( posItem != NULL )

{

pItem = (CTestContainer98Item\*)pDoc->GetNextItem( posItem );

dwHitResult = pItem->HitTest( point );

if( dwHitResult == HITRESULT\_HIT )

{

// We hit the control itself.

\*pdwHitType = HITTYPE\_OBJECT;

return( pItem );

}

else

{

// We missed the control, so check the grab border that we added to

// it.

rectBorder = pItem->m\_rect;

rectBorder.InflateRect( MARGIN\_PIXELS, MARGIN\_PIXELS );

if( rectBorder.PtInRect( point ) && !pItem->m\_rect.PtInRect( point ) )

{

// The point is outside the item's bounding rectangle, but inside

// the border rect.

\*pdwHitType = HITTYPE\_BORDER;

return( pItem );

}

}

}

return( NULL );

}

void CTestContainer98View::SetMultiSelect( CTestContainer98Item\* pItem,

BOOL tSelect )

{

POSITION posItem;

ENSURE( pItem != NULL );

posItem = m\_lpSelectedItems.Find( pItem );

if( tSelect )

{

if( posItem != NULL )

{

// Already selected

return;

}

else

{

m\_lpSelectedItems.AddHead( pItem );

if( m\_lpSelectedItems.GetCount() == 1 )

{

ASSERT( m\_pSelection == NULL );

m\_pSelection = pItem;

ENSURE( m\_pSelection != NULL );

m\_pSelection->OnSelect();

}

else

{

if( m\_pSelection != NULL )

{

OnUpdate( NULL, HINT\_UPDATE\_ITEM, m\_pSelection );

m\_pSelection->OnDeselect();

m\_pSelection = NULL;

}

}

OnUpdate( NULL, HINT\_UPDATE\_ITEM, pItem );

}

}

else

{

if( posItem == NULL )

{

// Already not selected

return;

}

else

{

if( m\_pSelection == pItem )

{

ASSERT( m\_lpSelectedItems.GetCount() == 1 );

m\_pSelection->OnDeselect();

m\_pSelection = NULL;

}

m\_lpSelectedItems.RemoveAt( posItem );

if( m\_lpSelectedItems.GetCount() == 1 )

{

ASSERT( m\_pSelection == NULL );

m\_pSelection = m\_lpSelectedItems.GetHead();

m\_pSelection->OnSelect();

OnUpdate( NULL, HINT\_UPDATE\_ITEM, m\_pSelection );

}

OnUpdate( NULL, HINT\_UPDATE\_ITEM, pItem );

}

}

}

void CTestContainer98View::ToggleMultiSelect( CTestContainer98Item\* pItem )

{

POSITION posItem;

ENSURE( pItem != NULL );

posItem = m\_lpSelectedItems.Find( pItem );

if( posItem == NULL )

{

m\_lpSelectedItems.AddHead( pItem );

if( m\_lpSelectedItems.GetCount() == 1 )

{

ASSERT( m\_pSelection == NULL );

m\_pSelection = pItem;

m\_pSelection->OnSelect();

}

else

{

if( m\_pSelection != NULL )

{

m\_pSelection->OnDeselect();

m\_pSelection = NULL;

}

}

}

else

{

if( m\_pSelection == pItem )

{

ASSERT( m\_lpSelectedItems.GetCount() == 1 );

m\_pSelection->OnDeselect();

m\_pSelection = NULL;

}

m\_lpSelectedItems.RemoveAt( posItem );

if( m\_lpSelectedItems.GetCount() == 1 )

{

ASSERT( m\_pSelection == NULL );

m\_pSelection = m\_lpSelectedItems.GetHead();

m\_pSelection->OnSelect();

}

}

OnUpdate( NULL, HINT\_UPDATE\_ITEM, pItem );

}

void CTestContainer98View::SetSelection( CTestContainer98Item\* pItem )

{

CTestContainer98Item\* pSelectedItem;

if( (pItem != NULL ) && (m\_pSelection == pItem) )

{

// No change in selection

return;

}

if( m\_pSelection != NULL )

{

m\_pSelection->OnDeselect();

m\_pSelection = NULL;

}

while( !m\_lpSelectedItems.IsEmpty() )

{

pSelectedItem = m\_lpSelectedItems.RemoveHead();

if( pSelectedItem != pItem )

{

OnUpdate( NULL, HINT\_UPDATE\_ITEM, pSelectedItem );

}

}

if( pItem != NULL )

{

m\_lpSelectedItems.AddHead( pItem );

m\_pSelection = pItem;

m\_pSelection->OnSelect();

OnUpdate( NULL, HINT\_UPDATE\_ITEM, pItem );

}

}

BOOL CTestContainer98View::HandleWindowlessKeyboardMessage()

{

if( m\_pItemWithFocus != NULL )

{

if( m\_pItemWithFocus->IsWindowless() )

{

m\_pItemWithFocus->OnWindowMessage( GetCurrentMessage() );

return( TRUE );

}

}

return( FALSE );

}

BOOL CTestContainer98View::HandlePointerInactiveMove( CPoint point,

DWORD dwKeyState )

{

CTestContainer98Item\* pItemHit;

CRect rect;

BOOL tHandled;

DWORD dwPolicy;

MSG msg;

DWORD dwHitType;

if( !GetDocument()->UseIPointerInactive() )

{

return( FALSE );

}

tHandled = FALSE;

pItemHit = HitTestItems( point, &dwHitType );

if( (dwHitType != HITTYPE\_OBJECT) || !pItemHit->SupportsIPointerInactive() )

{

pItemHit = NULL;

}

if( m\_pItemToDeactivateOnLeave != NULL )

{

// We have an active object that wants to be deactivated as soon as the

// mouse leaves its extents.

if( m\_pItemToDeactivateOnLeave != pItemHit )

{

TCTrace( TRACELEVEL\_NORMAL,

L"Deactivating item because of POINTERINACTIVE\_DEACTIVATEONLEAVE.\n" );

m\_pItemToDeactivateOnLeave->Deactivate();

m\_pItemToDeactivateOnLeave = NULL;

}

}

if( pItemHit != NULL )

{

if( !pItemHit->IsInPlaceActive() )

{

// The mouse was over an inactive control that wants to receive mouse

// pointer messages, so send the message to that control first.

dwPolicy = pItemHit->OnPointerInactiveMove( point, dwKeyState );

if( dwPolicy&POINTERINACTIVE\_ACTIVATEONENTRY )

{

msg = \*GetCurrentMessage();

TCTrace( TRACELEVEL\_NORMAL,

L"Activating item because of POINTERINACTIVE\_ACTIVATEONENTRY.\n" );

pItemHit->Activate( OLEIVERB\_SHOW, this, &msg );

if( dwPolicy&POINTERINACTIVE\_DEACTIVATEONLEAVE )

{

// The control wants to be deactivated as soon as the mouse

// leaves its extents.

m\_pItemToDeactivateOnLeave = pItemHit;

}

}

tHandled = TRUE;

}

}

else

{

tHandled = FALSE;

}

return( tHandled );

}

BOOL CTestContainer98View::HandlePointerInactiveSetCursor(

DWORD dwMouseMessage )

{

CTestContainer98Item\* pItemHit;

CRect rect;

BOOL tHandled;

DWORD dwPolicy;

MSG msg;

CPoint point;

DWORD dwHitType;

if( !GetDocument()->UseIPointerInactive() )

{

return( FALSE );

}

point = m\_ptLastMousePosition;

pItemHit = HitTestItems( point, &dwHitType );

if( (dwHitType != HITTYPE\_OBJECT) || !pItemHit->SupportsIPointerInactive() )

{

pItemHit = NULL;

}

if( m\_pItemToDeactivateOnLeave != NULL )

{

// If the pointer is still over the same control, then we really haven't

// left it.

if( m\_pItemToDeactivateOnLeave != pItemHit )

{

TCTrace( TRACELEVEL\_NORMAL,

L"Deactivating item because of POINTERINACTIVE\_DEACTIVATEONLEAVE.\n" );

m\_pItemToDeactivateOnLeave->Deactivate();

m\_pItemToDeactivateOnLeave = NULL;

}

}

tHandled = FALSE;

if( pItemHit != NULL )

{

if( !pItemHit->IsInPlaceActive() )

{

// The mouse was over an inactive control that wants to receive mouse

// pointer messages, so send the message to that control first.

dwPolicy = pItemHit->OnPointerInactiveSetCursor( point,

dwMouseMessage, &tHandled );

if( dwPolicy&POINTERINACTIVE\_ACTIVATEONENTRY )

{

msg = \*GetCurrentMessage();

TCTrace( TRACELEVEL\_NORMAL,

L"Activating item because of POINTERINACTIVE\_ACTIVATEONENTRY.\n" );

pItemHit->Activate( OLEIVERB\_SHOW, this, &msg );

if( dwPolicy&POINTERINACTIVE\_DEACTIVATEONLEAVE )

{

m\_pItemToDeactivateOnLeave = pItemHit;

}

tHandled = TRUE;

}

}

}

return( tHandled );

}

BOOL CTestContainer98View::HandlePointerInactiveDragOver( CPoint point )

{

CTestContainer98Item\* pItemHit;

DWORD dwHitType;

DWORD dwPolicy;

BOOL tHandled;

if( !GetDocument()->UseIPointerInactive() )

{

return( FALSE );

}

pItemHit = HitTestItems( point, &dwHitType );

if( (dwHitType != HITTYPE\_OBJECT) || !pItemHit->SupportsIPointerInactive() )

{

pItemHit = NULL;

}

tHandled = FALSE;

if( pItemHit != NULL )

{

if( !pItemHit->IsInPlaceActive() )

{

dwPolicy = pItemHit->GetActivationPolicy();

if( dwPolicy&POINTERINACTIVE\_ACTIVATEONDRAG )

{

TCTrace( TRACELEVEL\_NORMAL,

L"Activating item because of POINTERINACTIVE\_ACTIVATEONDRAG.\n" );

pItemHit->Activate( OLEIVERB\_SHOW, this );

}

}

}

return( tHandled );

BOOL CTestContainer98View::HandleWindowlessDragEnter(

COleDataObject\* pDataObject, DWORD dwKeyState, CPoint point,

DWORD\* pdwDropEffect )

{

CTestContainer98Item\* pItemHit;

DWORD dwHitType;

BOOL tHandled;

ASSERT( m\_pWindowlessDropTarget == NULL );

pItemHit = HitTestItems( point, &dwHitType );

if( (dwHitType != HITTYPE\_OBJECT) || !pItemHit->IsWindowless() )

{

pItemHit = NULL;

}

tHandled = FALSE;

if( pItemHit != NULL )

{

tHandled = pItemHit->OnWindowlessDragEnter( pDataObject, dwKeyState,

point, pdwDropEffect );

if( tHandled )

{

// Remember which item we're over.

m\_pWindowlessDropTarget = pItemHit;

}

}

return( tHandled );

}

BOOL CTestContainer98View::HandleWindowlessDragOver(

COleDataObject\* pDataObject, DWORD dwKeyState, CPoint point,

DWORD\* pdwDropEffect )

{

CTestContainer98Item\* pItemHit;

DWORD dwHitType;

BOOL tHandled;

pItemHit = HitTestItems( point, &dwHitType );

if( (dwHitType != HITTYPE\_OBJECT) || !pItemHit->IsWindowless() )

{

pItemHit = NULL;

}

tHandled = FALSE;

if( m\_pWindowlessDropTarget != NULL )

{

// The cursor was previously over a windowless control.

if( m\_pWindowlessDropTarget != pItemHit )

{

// The cursor left that windowless control.

m\_pWindowlessDropTarget->OnWindowlessDragLeave();

m\_pWindowlessDropTarget = NULL;

}

}

if( pItemHit != NULL )

{

// The cursor is now over a windowless control.

if( pItemHit != m\_pWindowlessDropTarget )

{

// The cursor entered a new windowless control.

ASSERT( m\_pWindowlessDropTarget == NULL );

tHandled = pItemHit->OnWindowlessDragEnter( pDataObject, dwKeyState,

point, pdwDropEffect );

if( tHandled )

{

m\_pWindowlessDropTarget = pItemHit;

}

}

else

{

// The cursor is still over the same windowless control.

pItemHit->OnWindowlessDragOver( dwKeyState, point, pdwDropEffect );

tHandled = TRUE;

}

}

return( tHandled );

}

void CTestContainer98View::HandleWindowlessDragLeave()

{

if( m\_pWindowlessDropTarget != NULL )

{

m\_pWindowlessDropTarget->OnWindowlessDragLeave();

m\_pWindowlessDropTarget = NULL;

}

}

BOOL CTestContainer98View::HandleWindowlessDrop( COleDataObject\* pDataObject,

DWORD dwKeyState, CPoint point, DWORD\* pdwEffect )

{

if( m\_pWindowlessDropTarget != NULL )

{

m\_pWindowlessDropTarget->OnWindowlessDrop( pDataObject, dwKeyState,

point, pdwEffect );

m\_pWindowlessDropTarget = NULL;

return( TRUE );

}

else

{

return( FALSE );

}

}

BOOL CTestContainer98View::HandleWindowlessMouseMessage( CPoint point )

{

CTestContainer98Item\* pItemHit;

DWORD dwHitType;

BOOL tHandled;

CRect rect;

// First, see if a windowless control has mouse capture.

if( m\_pItemWithCapture != NULL )

{

m\_pItemWithCapture->OnWindowMessage( GetCurrentMessage() );

return( TRUE );

}

// Now, see if we hit a windowless control.

pItemHit = HitTestItems( point, &dwHitType );

if( (dwHitType != HITTYPE\_OBJECT) || !pItemHit->IsWindowless() )

{

pItemHit = NULL;

}

if( pItemHit != NULL )

{

// The mouse was over a windowless control, so send the message to that

// control first.

tHandled = pItemHit->OnWindowMessage( GetCurrentMessage() );

if( !tHandled )

{

// The windowless control didn't do anything, but we don't want the

// container to do anything either.

{

Default();

}

}

return( TRUE );

}

return( FALSE );

}

void CTestContainer98View::OnLButtonDown( UINT nFlags, CPoint point )

{

BOOL tHandled;

CTestContainer98Item\* pItemHit;

CTestContainer98Item\* pItem;

POSITION posItem;

DWORD dwHitType;

int eHitType;

MSG msg;

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

pItemHit = HitTestItems( point, &dwHitType );

if( nFlags&MK\_CONTROL )

{

if( pItemHit != NULL )

{

ToggleMultiSelect( pItemHit );

}

}

else

{

if( m\_lpSelectedItems.Find( pItemHit ) == NULL )

{

SetSelection( pItemHit );

}

}

if( (pItemHit != NULL) && (dwHitType != HITTYPE\_BORDER) )

{

if( !pItemHit->IsInPlaceActive() )

{

msg = \*GetCurrentMessage();

pItemHit->Activate( OLEIVERB\_SHOW, this, &msg );

}

}

if( !m\_lpSelectedItems.IsEmpty() )

{

if( dwHitType == HITTYPE\_BORDER )

{

CRectTracker tracker;

CPoint ptStart;

CSize sizeDisplacement;

CRect rectItem;

SetupSelectionTracker( &tracker );

ptStart = tracker.m\_rect.TopLeft();

eHitType = tracker.HitTest( point );

if( eHitType == CRectTracker::hitMiddle )

{

// Drag the selection

if( tracker.Track( this, point ) )

{

posItem = m\_lpSelectedItems.GetHeadPosition();

while( posItem != NULL )

{

pItem = m\_lpSelectedItems.GetNext( posItem );

pItem->InvalidateItem();

}

sizeDisplacement = tracker.m\_rect.TopLeft()-ptStart;

posItem = m\_lpSelectedItems.GetHeadPosition();

while( posItem != NULL )

{

pItem = m\_lpSelectedItems.GetNext( posItem );

rectItem = pItem->m\_rect;

rectItem.OffsetRect( sizeDisplacement );

pItem->Move( rectItem );

pItem->InvalidateItem();

}

GetDocument()->SetModifiedFlag();

}

}

else

{

if( (eHitType != CRectTracker::hitNothing) && (m\_pSelection !=

NULL) )

{

// Resize the item.

if( tracker.Track( this, point ) )

{

m\_pSelection->InvalidateItem();

}

tracker.m\_rect.DeflateRect( MARGIN\_PIXELS, MARGIN\_PIXELS );

m\_pSelection->Move( tracker.m\_rect );

m\_pSelection->InvalidateItem();

GetDocument()->SetModifiedFlag();

}

}

}

}

else

{

CRectTracker tracker;

CRect rectIntersection;

CTestContainer98Doc\* pDoc;

if( tracker.TrackRubberBand( this, point, TRUE ) )

{

tracker.m\_rect.NormalizeRect();

pDoc = GetDocument();

posItem = pDoc->GetStartPosition();

while( posItem != NULL )

{

pItem = STATIC\_DOWNCAST( CTestContainer98Item, pDoc->GetNextItem(

posItem ) );

if( rectIntersection.IntersectRect( tracker.m\_rect,

pItem->m\_rect ) )

{

SetMultiSelect( pItem, TRUE );

}

}

}

}

}

void CTestContainer98View::SetupSelectionTracker( CRectTracker\* pTracker )

{

POSITION posItem;

CTestContainer98Item\* pItem;

ASSERT( !m\_lpSelectedItems.IsEmpty() );

if( m\_pSelection != NULL )

{

// Only one item is selected, so use its tracker.

SetupTracker( m\_pSelection, pTracker );

}

else

{

pTracker->m\_rect.SetRectEmpty();

posItem = m\_lpSelectedItems.GetHeadPosition();

while( posItem != NULL )

{

pItem = m\_lpSelectedItems.GetNext( posItem );

pTracker->m\_rect.UnionRect( pTracker->m\_rect, pItem->m\_rect );

}

pTracker->m\_rect.InflateRect( MARGIN\_PIXELS, MARGIN\_PIXELS );

pTracker->m\_nHandleSize = MARGIN\_PIXELS+1;

pTracker->m\_sizeMin.cx = 8;

pTracker->m\_sizeMin.cy = 8;

pTracker->m\_nStyle = CRectTracker::solidLine;

}

}

void CTestContainer98View::SetupTracker( CTestContainer98Item\* pItem,

CRectTracker\* pTracker )

{

pTracker->m\_rect = pItem->m\_rect;

pTracker->m\_rect.InflateRect( MARGIN\_PIXELS, MARGIN\_PIXELS );

pTracker->m\_nHandleSize = MARGIN\_PIXELS+1;

pTracker->m\_sizeMin.cx = 8;

pTracker->m\_sizeMin.cy = 8;

pTracker->m\_nStyle = 0;

if( m\_lpSelectedItems.Find( pItem ) != NULL )

{

pTracker->m\_nStyle |= CRectTracker::resizeInside;

}

pTracker->m\_nStyle |= CRectTracker::solidLine;

if( (pItem->GetItemState() == COleClientItem::openState) ||

(pItem->GetItemState() == COleClientItem::activeUIState) )

{

pTracker->m\_nStyle |= CRectTracker::hatchInside;

}

}

void CTestContainer98View::OnLButtonDblClk( UINT nFlags, CPoint point )

{

LONG iVerb;

BOOL tHandled;

CTestContainer98Item\* pItemHit;

MSG msg;

DWORD dwHitType;

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

pItemHit = HitTestItems( point, &dwHitType );

SetSelection( pItemHit );

if( pItemHit != NULL )

{

if( !GetDocument()->GetUserMode() && pItemHit->IsInsideOut() &&

!pItemHit->IsUIActive() )

{

// UIActivate the selected object.

msg = \*GetCurrentMessage();

pItemHit->DoVerb( OLEIVERB\_UIACTIVATE, this, &msg );

}

}

Invalidate();

if( m\_pSelection != NULL )

{

if( GetKeyState( VK\_CONTROL ) < 0 )

{

iVerb = OLEIVERB\_OPEN;

}

else

{

iVerb = OLEIVERB\_PRIMARY;

}

m\_pSelection->DoVerb( iVerb, this );

}

CView::OnLButtonDblClk( nFlags, point );

}

HRESULT CTestContainer98View::OnWindowlessDefWindowMessage( UINT nMessage,

WPARAM wParam, LPARAM lParam, LRESULT\* plResult )

{

const MSG\* pMessage;

(void)nMessage;

(void)wParam;

(void)lParam;

pMessage = GetCurrentMessage();

ASSERT( pMessage->message == nMessage );

ASSERT( pMessage->wParam == wParam );

ASSERT( pMessage->lParam == lParam );

\*plResult = Default();

return( S\_OK );

}

BOOL CTestContainer98View::OnSetCursor( CWnd\* pWnd, UINT nHitTest,

UINT message )

{

CPoint pt;

DWORD dwHitType;

CRectTracker tracker;

BOOL tHandled;

CTestContainer98Item\* pItemHit;

// First, give the any inactive objects underneath the cursor a chance to

// set the cursor and/or activate.

tHandled = HandlePointerInactiveSetCursor( message );

if( tHandled )

{

return( TRUE );

}

// Now, see if a windowless control should be handling this message.

tHandled = HandleWindowlessMouseMessage( m\_ptLastMousePosition );

if( tHandled )

{

return( FALSE );

}

// None of the controls handled the message, so set the cursor ourselves.

// The only interesting thing we do is let the tracker for the current

// selection set the cursor to show resize/move arrows.

if( pWnd == this )

{

GetCursorPos( &pt );

ScreenToClient( &pt );

pItemHit = HitTestItems( pt, &dwHitType );

if( (pItemHit != NULL) && (dwHitType == HITTYPE\_BORDER) &&

(m\_lpSelectedItems.Find( pItemHit ) != NULL) )

{

SetupTracker( pItemHit, &tracker );

if( tracker.SetCursor( this, nHitTest ) )

{

return( TRUE );

}

}

}

return( CView::OnSetCursor( pWnd, nHitTest, message ) );

}

void CTestContainer98View::OnEditClear()

{

POSITION posItem;

CTestContainer98Item\* pItem;

ASSERT( !m\_lpSelectedItems.IsEmpty() );

posItem = m\_lpSelectedItems.GetHeadPosition();

while( posItem != NULL )

{

pItem = m\_lpSelectedItems.GetNext( posItem );

pItem->Delete();

}

ClearSelection();

GetDocument()->UpdateAllViews( NULL );

}

void CTestContainer98View::OnUpdateEditClear( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( !m\_lpSelectedItems.IsEmpty() );

}

void CTestContainer98View::OnUpdate( CView\* pSender, LPARAM lHint,

CObject\* pHint )

{

(void)pSender;

switch( lHint )

{

case HINT\_UPDATE\_WINDOW:

Invalidate();

break;

case HINT\_UPDATE\_ITEM:

{

CRectTracker tracker;

CRect rect;

SetupTracker( (CTestContainer98Item\*)pHint, &tracker );

tracker.GetTrueRect( &rect );

InvalidateRect( rect );

}

break;

}

}

void CTestContainer98View::OnRButtonDown( UINT nFlags, CPoint point )

{

CTestContainer98Item\* pItemHit;

BOOL tHandled;

DWORD dwHitType;

MSG msg;

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

pItemHit = HitTestItems( point, &dwHitType );

if( !(nFlags&MK\_CONTROL) )

{

if( (pItemHit != NULL) && (m\_lpSelectedItems.Find( pItemHit ) == NULL) )

{

SetSelection( pItemHit );

}

}

if( (pItemHit != NULL) && (dwHitType != HITTYPE\_BORDER) )

{

if( !pItemHit->IsInPlaceActive() )

{

msg = \*GetCurrentMessage();

pItemHit->Activate( OLEIVERB\_SHOW, this, &msg );

}

}

}

BOOL CTestContainer98View::PreTranslateMessage( MSG\* pMsg )

{

CTestContainer98Item\* pItem;

CTestContainer98Doc\* pDoc;

MSG msg;

IOleInPlaceActiveObjectPtr pInPlaceActiveObject;

HRESULT hResult;

POSITION posItem;

CTestContainer98Item\* pCurrentItem;

pDoc = GetDocument();

pCurrentItem = NULL;

if( m\_pItemWithFocus != NULL )

{

pCurrentItem = m\_pItemWithFocus;

}

else

{

// Try the UI active control

posItem = pDoc->GetStartPosition();

while( (posItem != NULL) && (pCurrentItem == NULL) )

{

pItem = (CTestContainer98Item\*)pDoc->GetNextItem( posItem );

if( pItem->IsUIActive() )

{ pCurrentItem = pItem;

}

}

}

if( ((pMsg->message >= WM\_KEYFIRST) && (pMsg->message <= WM\_KEYLAST)) ||

((pMsg->message >= WM\_MOUSEFIRST) && (pMsg->message <= WM\_MOUSELAST)) )

{

// Give the item with focus first crack at keyboard and mouse messages

if( pCurrentItem != NULL )

{

pInPlaceActiveObject = pCurrentItem->m\_lpObject;

if( pInPlaceActiveObject != NULL )

{

hResult = pInPlaceActiveObject->TranslateAccelerator( pMsg );

if( hResult == S\_OK )

{

return( TRUE );

}

}

}

}

if( pMsg->message == WM\_KEYDOWN )

{

if( LOWORD( pMsg->wParam ) == VK\_TAB )

{

pItem = pDoc->GetNextTabItem( pCurrentItem );

if( pItem != NULL )

{

if( pItem->HasUIActiveState() )

{

if( !pItem->IsUIActive() )

{

msg = \*pMsg;

pItem->DoVerb( OLEIVERB\_UIACTIVATE, this, &msg );

}

}

if( pItem->ActsLikeButton() )

{

pItem->DisplayAsDefault( TRUE );

}

}

return( TRUE );

}

}

else if( pMsg->message == WM\_KEYUP )

{

if( LOWORD( pMsg->wParam ) == VK\_TAB )

{

return( TRUE );

}

}

return( CView::PreTranslateMessage( pMsg ) );

}

void CTestContainer98View::OnOptionsDesignMode()

{

CTestContainer98Doc\* pDoc;

pDoc = GetDocument();

pDoc->SetUserMode( !pDoc->GetUserMode() );

}

void CTestContainer98View::OnUpdateOptionsDesignMode( CCmdUI\* pCmdUI )

{

if( GetDocument()->GetUserMode() )

{

pCmdUI->SetCheck( 0 );

}

else

{

pCmdUI->SetCheck( 1 );

}

}

BOOL CTestContainer98View::OnDrop( COleDataObject\* pDataObject,

DROPEFFECT dropEffect, CPoint point )

{

DWORD dwDropEffect;

BOOL tHandled;

TRACE( "OnDrop()\n" );

dwDropEffect = dropEffect;

tHandled = HandleWindowlessDrop( pDataObject, m\_dwLastDropKeyState, point,

&dwDropEffect );

if( tHandled )

{

return( CView::OnDrop( pDataObject, dropEffect, point ) );

}

return( CView::OnDrop( pDataObject, dropEffect, point ) );

}

CTestContainer98Item\* CTestContainer98View::GetItemWithCapture() const

{

return( m\_pItemWithCapture );

}

CTestContainer98Item\* CTestContainer98View::GetItemWithFocus() const

{

return( m\_pItemWithFocus );

}

HRESULT CTestContainer98View::OnWindowlessReleaseCapture(

CTestContainer98Item\* pItem )

{

if( m\_pItemWithCapture == pItem )

{

m\_pItemWithCapture = NULL;

ReleaseCapture();

}

return( S\_OK );

}

HRESULT CTestContainer98View::OnWindowlessReleaseFocus()

{

ASSERT( m\_pItemWithFocus != NULL );

m\_pItemWithFocus = NULL;

return( S\_OK );

}

HRESULT CTestContainer98View::OnWindowlessSetCapture(

CTestContainer98Item\* pItem )

{

ASSERT( pItem != NULL );

// ASSERT( m\_pItemWithCapture == NULL );

SetCapture();

m\_pItemWithCapture = pItem;

return( S\_OK );

}

HRESULT CTestContainer98View::OnWindowlessSetFocus(

CTestContainer98Item\* pItem )

{

ASSERT( pItem != NULL );

TCTrace( TRACELEVEL\_NORMAL, L"Setting focus to windowless control.\n" );

SetFocus();

m\_pItemWithFocus = pItem;

return( S\_OK );

}

DROPEFFECT CTestContainer98View::OnDragEnter( COleDataObject\* pDataObject,

DWORD dwKeyState, CPoint point )

{

BOOL tHandled;

DWORD dwDropEffect;

TRACE( "OnDragEnter()\n" );

HandlePointerInactiveDragOver( point );

dwDropEffect = DROPEFFECT\_COPY|DROPEFFECT\_MOVE|DROPEFFECT\_LINK;

tHandled = HandleWindowlessDragEnter( pDataObject, dwKeyState, point,

&dwDropEffect );

if( tHandled )

{

return( dwDropEffect );

}

return( DROPEFFECT\_NONE );

}

DROPEFFECT CTestContainer98View::OnDragOver( COleDataObject\* pDataObject,

DWORD dwKeyState, CPoint point )

{

DWORD dwDropEffect;

BOOL tHandled;

m\_dwLastDropKeyState = dwKeyState;

HandlePointerInactiveDragOver( point );

dwDropEffect = DROPEFFECT\_COPY|DROPEFFECT\_MOVE|DROPEFFECT\_LINK;

tHandled = HandleWindowlessDragOver( pDataObject, dwKeyState, point,

&dwDropEffect );

if( tHandled )

{

return( dwDropEffect );

}

return( DROPEFFECT\_NONE );

}

int CTestContainer98View::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if( CView::OnCreate( lpCreateStruct ) == -1 )

{

return( -1 );

}

// Register drop target.

m\_dropTarget.Register( this );

return( 0 );

}

void CTestContainer98View::OnLButtonUp( UINT nFlags, CPoint point )

{

BOOL tHandled;

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

CView::OnLButtonUp( nFlags, point );

}

void CTestContainer98View::OnMouseMove( UINT nFlags, CPoint point )

{

BOOL tHandled;

m\_ptLastMousePosition = point;

tHandled = HandlePointerInactiveMove( point, nFlags );

if( tHandled )

{

return;

}

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

CView::OnMouseMove( nFlags, point );

}

void CTestContainer98View::OnRButtonDblClk( UINT nFlags, CPoint point )

{

BOOL tHandled;

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

CView::OnRButtonDblClk( nFlags, point );

}

void CTestContainer98View::OnRButtonUp( UINT nFlags, CPoint point )

{

BOOL tHandled;

tHandled = HandleWindowlessMouseMessage( point );

if( tHandled )

{

return;

}

CView::OnRButtonUp( nFlags, point );

}

void CTestContainer98View::OnCaptureChanged( CWnd \*pWnd )

{

// We're losing the capture, so the windowless item that has the capture

// loses it, too.

m\_pItemWithCapture = NULL;

CView::OnCaptureChanged( pWnd );

}

void CTestContainer98View::OnFocus( CTestContainer98Item\* pItem,

BOOL tGotFocus )

{

ASSERT( pItem != NULL );

if( tGotFocus )

{

m\_pItemWithFocus = pItem;

}

else

{

if( m\_pItemWithFocus == pItem )

{

m\_pItemWithFocus = NULL;

}

}

}

void CTestContainer98View::OnKillFocus( CWnd\* pNewWnd )

{

CView ::OnKillFocus( pNewWnd );

}

void CTestContainer98View::OnKeyDown( UINT nChar, UINT nRepCnt, UINT nFlags )

{

BOOL tHandled;

tHandled = HandleWindowlessKeyboardMessage();

if( tHandled )

{

return;

}

CView::OnKeyDown( nChar, nRepCnt, nFlags );

}

void CTestContainer98View::OnKeyUp( UINT nChar, UINT nRepCnt, UINT nFlags )

{

BOOL tHandled;

tHandled = HandleWindowlessKeyboardMessage();

if( tHandled )

{

return;

}

CView::OnKeyUp( nChar, nRepCnt, nFlags );

}

void CTestContainer98View::OnChar( UINT nChar, UINT nRepCnt, UINT nFlags )

{

BOOL tHandled;

tHandled = HandleWindowlessKeyboardMessage();

if( tHandled )

{

return;

}

CView::OnChar( nChar, nRepCnt, nFlags );

}

void CTestContainer98View::OnCancelMode()

{

BOOL tHandled;

CView::OnCancelMode();

tHandled = HandleWindowlessKeyboardMessage();

if( tHandled )

{

return;

}

}

void CTestContainer98View::OnControlSaveToStream()

{

int nResult;

CString strFilter;

LOAD\_STRING\_FROM\_RESOURCE(strFilter, IDS\_STREAM\_FILES);

CFileDialog dlg( FALSE, \_T( "TSM" ), NULL, OFN\_HIDEREADONLY|

OFN\_OVERWRITEPROMPT, strFilter);

HRESULT hResult;

IStreamPtr pStream;

LARGE\_INTEGER nDisplacement;

ULARGE\_INTEGER iSeekPtr;

ULONG nBytesLeft;

ULONG nBytesRead;

BYTE abData[512];

ENSURE( m\_pSelection != NULL );

nResult = (int)dlg.DoModal();

if( nResult != IDOK )

{

return;

}

hResult = CreateStreamOnHGlobal( NULL, TRUE, &pStream );

if( FAILED( hResult ) )

{

TRACE( "CreateStreamOnHGlobal() failed.\n" );

return;

}

hResult = m\_pSelection->SaveToStream( pStream );

if( FAILED( hResult ) )

{

CString strErrorMessage;

\_com\_error e( hResult );

LOAD\_STRING\_FROM\_RESOURCE(strErrorMessage, IDS\_SAVETOSTREAMFAILED);

strErrorMessage += e.ErrorMessage();

AfxMessageBox( strErrorMessage );

return;

}

try

{

CFile file( dlg.GetPathName(), CFile::modeCreate|CFile::modeWrite|

CFile::shareExclusive );

nDisplacement.QuadPart = 0;

pStream->Seek( nDisplacement, STREAM\_SEEK\_CUR, &iSeekPtr );

nBytesLeft = ULONG( iSeekPtr.QuadPart );

pStream->Seek( nDisplacement, STREAM\_SEEK\_SET, NULL );

while( nBytesLeft > 0 )

{

pStream->Read( abData, min( nBytesLeft, sizeof( abData ) ),

&nBytesRead );

ASSERT( nBytesRead > 0 );

file.Write( abData, nBytesRead );

nBytesLeft -= nBytesRead;

}

}

catch( CFileException\* pException )

{

pException->Delete();

AfxMessageBox( IDS\_ERRORWRITINGTOFILE );

}

}

void CTestContainer98View::OnEditInsertFromStream()

{

int nResult;

HRESULT hResult;

IStreamPtr pStream;

LARGE\_INTEGER nDisplacement;

ULONG nBytesLeft;

ULONG nBytesToRead;

ULONGLONG nFileLength;

CTestContainer98Doc\* pDoc;

CTestContainer98Item\* pItem;

CLSID clsid;

BYTE abData[512];

CString strFilter;

LOAD\_STRING\_FROM\_RESOURCE(strFilter, IDS\_STREAM\_FILES);

CFileDialog dlg( TRUE, \_T( "TSM" ), NULL, OFN\_HIDEREADONLY|

OFN\_OVERWRITEPROMPT, strFilter );

nResult = (int)dlg.DoModal();

if( nResult != IDOK )

{

return;

}

hResult = CreateStreamOnHGlobal( NULL, TRUE, &pStream );

if( FAILED( hResult ) )

{

TRACE( "CreateStreamOnHGlobal() failed.\n" );

return;

}

try

{

CFile file( dlg.GetPathName(), CFile::modeRead|CFile::shareDenyWrite );

nFileLength = file.GetLength();

if( nFileLength > INT\_MAX )

{

AfxThrowOleException( E\_FAIL );

}

nBytesLeft = ULONG( nFileLength );

while( nBytesLeft > 0 )

{

nBytesToRead = min( nBytesLeft, sizeof( abData ) );

file.Read( abData, nBytesToRead );

pStream->Write( abData, nBytesToRead, NULL );

nBytesLeft -= nBytesToRead;

}

}

catch( CException\* pException )

{

pException->Delete();

AfxMessageBox( IDS\_ERRORREADINGFROMFILE );

return;

}

nDisplacement.QuadPart = 0;

pStream->Seek( nDisplacement, STREAM\_SEEK\_SET, NULL );

BeginWaitCursor();

pItem = NULL;

try

{

hResult = ReadClassStm( pStream, &clsid );

if( FAILED( hResult ) )

{

AfxThrowOleException( hResult );

}

pDoc = GetDocument();

pItem = pDoc->AddControl( clsid, NULL, IID\_IStream, pStream );

}

catch( COleException\* pException )

{

CString strErrorMessage;

\_com\_error error( pException->m\_sc );

pException->Delete();

if( pItem != NULL )

{

ASSERT\_VALID( pItem );

pItem->Delete();

}

AfxFormatString1( strErrorMessage, IDS\_CREATEFAILED,

error.ErrorMessage() );

AfxMessageBox( strErrorMessage );

}

catch( CException\* pException )

{

pException->Delete();

AfxMessageBox( IDP\_FAILED\_TO\_CREATE );

}

EndWaitCursor();

}

void CTestContainer98View::OnDragLeave()

{

TRACE( "OnDragLeave()\n" );

HandleWindowlessDragLeave();

CView::OnDragLeave();

}

void CTestContainer98View::OnControlSaveToPropertyBag()

{

CPropertyBag\* pPropertyBagObject;

IPropertyBagPtr pPropertyBag;

HRESULT hResult;

#pragma warning(suppress: 6014) // unexpected prefast memory leak warning

pPropertyBagObject = new CPropertyBag;

if( pPropertyBagObject == NULL )

{

return;

}

hResult = pPropertyBagObject->QueryInterface( IID\_IPropertyBag,

(void\*\*)&pPropertyBag );

ASSERT( SUCCEEDED( hResult ) );

hResult = m\_pSelection->SaveToPropertyBag( pPropertyBag );

if( FAILED( hResult ) )

{

CString strErrorMessage;

\_com\_error e( hResult );

LOAD\_STRING\_FROM\_RESOURCE(strErrorMessage, IDS\_SAVETOPROPERTYBAGFAILED);

strErrorMessage += e.ErrorMessage();

AfxMessageBox( strErrorMessage );

return;

}

CPropertyBagDlg dlg( pPropertyBagObject );

dlg.DoModal();

}

void CTestContainer98View::OnUpdateControlSaveToPropertyBag( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( m\_pSelection != NULL );

}

void CTestContainer98View::OnUpdateControlSaveToStream( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( m\_pSelection != NULL );

}

void CTestContainer98View::OnControlInvokeMethods()

{

ENSURE( m\_pSelection != NULL );

m\_pSelection->InvokeMethods();

}

void CTestContainer98View::OnUpdateControlInvokeMethods( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( m\_pSelection != NULL );

}

void CTestContainer98View::OnControlLogging()

{

ENSURE( m\_pSelection != NULL );

m\_pSelection->SetLoggingOptions();

}

void CTestContainer98View::OnUpdateControlLogging( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( m\_pSelection != NULL );

}

void CTestContainer98View::OnControlActivate()

{

ENSURE( m\_pSelection != NULL );

m\_pSelection->Activate( OLEIVERB\_INPLACEACTIVATE, this );

}

void CTestContainer98View::OnControlDeactivate()

{

ENSURE( m\_pSelection != NULL );

m\_pSelection->Deactivate();

}

void CTestContainer98View::OnControlUIActivate()

{

ENSURE( m\_pSelection != NULL );

m\_pSelection->Activate( OLEIVERB\_UIACTIVATE, this );

}

void CTestContainer98View::OnControlUIDeactivate()

{

ENSURE( m\_pSelection != NULL );

m\_pSelection->DeactivateUI();

}

void CTestContainer98View::OnUpdateControlUIDeactivate( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( (m\_pSelection != NULL) && (m\_pSelection->IsUIActive()) );

}

void CTestContainer98View::OnUpdateControlUIActivate( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( (m\_pSelection != NULL) && (!m\_pSelection->IsUIActive()) );

}

void CTestContainer98View::OnUpdateControlActivate( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( (m\_pSelection != NULL) &&

(!m\_pSelection->IsInPlaceActive()) );

}

void CTestContainer98View::OnUpdateControlDeactivate( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( (m\_pSelection != NULL) &&

(m\_pSelection->IsInPlaceActive()) );

}

void CTestContainer98View::OnEditBringToFront()

{

POSITION posItem;

CTestContainer98Item\* pItem;

ASSERT( !m\_lpSelectedItems.IsEmpty() );

posItem = m\_lpSelectedItems.GetHeadPosition();

while( posItem != NULL )

{

pItem = m\_lpSelectedItems.GetNext( posItem );

GetDocument()->BringToFront( pItem );

}

Invalidate();

}

void CTestContainer98View::OnUpdateEditBringToFront( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( !m\_lpSelectedItems.IsEmpty() );

}

void CTestContainer98View::OnEditSendToBack()

{

POSITION posItem;

CTestContainer98Item\* pItem;

ASSERT( !m\_lpSelectedItems.IsEmpty() );

posItem = m\_lpSelectedItems.GetHeadPosition();

while( posItem != NULL )

{

pItem = m\_lpSelectedItems.GetNext( posItem );

GetDocument()->SendToBack( pItem );

}

Invalidate();

}

void CTestContainer98View::OnUpdateEditSendToBack( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( !m\_lpSelectedItems.IsEmpty() );

}

void CTestContainer98View::OnUpdateControlSaveToStorage( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( m\_lpSelectedItems.GetCount() == 1 );

}

void CTestContainer98View::OnEditInsertControlFromStorage()

{

CTestContainer98Item\* pItem;

CTestContainer98Doc\* pDoc;

int nResult;

IStoragePtr pStorage;

HRESULT hResult;

CLSID clsid;

CString strMessage;

CString strFilter;

LOAD\_STRING\_FROM\_RESOURCE(strFilter, IDS\_STORAGE\_FILES);

CFileDialog dlg( TRUE, \_T( "TSG" ), NULL, OFN\_HIDEREADONLY|

OFN\_OVERWRITEPROMPT, strFilter);

nResult = (int)dlg.DoModal();

if( nResult != IDOK )

{

return;

}

hResult = StgOpenStorage( CT2COLE( dlg.GetPathName() ), NULL, STGM\_READ|

STGM\_SHARE\_DENY\_WRITE, NULL, 0, &pStorage );

if( FAILED( hResult ) )

{

\_com\_error error( hResult );

AfxFormatString1( strMessage, IDS\_OPENSTORAGEFAILED,

error.ErrorMessage() );

AfxMessageBox( strMessage );

return;

}

ReadClassStg( pStorage, &clsid );

pDoc = GetDocument();

pItem = NULL;

try

{

pItem = pDoc->AddControl( clsid, NULL, IID\_IStorage, pStorage );

}

catch( COleException\* pException )

{

\_com\_error error( pException->m\_sc );

pException->Delete();

AfxFormatString1( strMessage, IDS\_LOADCONTROLFAILED,

error.ErrorMessage() );

AfxMessageBox( strMessage );

}

catch( CException\* pException )

{

\_com\_error error( E\_FAIL );

pException->Delete();

AfxFormatString1( strMessage, IDS\_LOADCONTROLFAILED,

error.ErrorMessage() );

AfxMessageBox( strMessage );

}

}

void CTestContainer98View::OnUpdateEditSelectAll( CCmdUI\* pCmdUI )

{

CTestContainer98Doc\* pDoc;

pDoc = GetDocument();

pCmdUI->Enable( pDoc->GetStartPosition() != NULL );

}

void CTestContainer98View::OnControlDrawMetaFile()

{

IUnknownPtr pObject;

CMetaFileWnd\* pWnd;

CString strWindowTitle;

ASSERT( m\_lpSelectedItems.GetCount() == 1 );

ENSURE( m\_pSelection != NULL );

pObject = m\_pSelection->m\_lpObject;

LOAD\_STRING\_FROM\_RESOURCE( strWindowTitle, IDS\_METAFILEWINDOWTITLE );

#pragma warning(suppress: 6014) // pWnd will be freed when the window is closed.

pWnd = new CMetaFileWnd;

pWnd->Create( NULL, strWindowTitle );

pWnd->SetObject( pObject );

pWnd->ShowWindow( SW\_SHOW );

}

void CTestContainer98View::OnUpdateControlDrawMetaFile( CCmdUI\* pCmdUI )

{

pCmdUI->Enable( m\_lpSelectedItems.GetCount() == 1 );

}

static int sig\_ignored(struct task\_struct \*t, int sig)

{

void \_\_user \* handler;

/\*

\* Tracers always want to know about signals..

\*/

if (t->ptrace & PT\_PTRACED)

return 0;

/\*

\* Blocked signals are never ignored, since the

\* signal handler may change by the time it is

\* unblocked.

\*/

if (sigismember(&t->blocked, sig))

return 0;

/\* Is it explicitly or implicitly ignored? \*/

handler = t->sighand->action[sig-1].sa.sa\_handler;

return handler == SIG\_IGN ||

(handler == SIG\_DFL && sig\_kernel\_ignore(sig));

}

/\*

\* Re-calculate pending state from the set of locally pending

\* signals, globally pending signals, and blocked signals.

\*/

static inline int has\_pending\_signals(sigset\_t \*signal, sigset\_t \*blocked)

{

unsigned long ready;

long i;

switch (\_NSIG\_WORDS) {

default:

for (i = \_NSIG\_WORDS, ready = 0; --i >= 0 ;)

ready |= signal->sig[i] &~ blocked->sig[i];

break;

case 4: ready = signal->sig[3] &~ blocked->sig[3];

ready |= signal->sig[2] &~ blocked->sig[2];

ready |= signal->sig[1] &~ blocked->sig[1];

ready |= signal->sig[0] &~ blocked->sig[0];

break;

case 2: ready = signal->sig[1] &~ blocked->sig[1];

ready |= signal->sig[0] &~ blocked->sig[0];

break;

case 1: ready = signal->sig[0] &~ blocked->sig[0];

}

return ready != 0;

}

#define PENDING(p,b) has\_pending\_signals(&(p)->signal, (b))

fastcall void recalc\_sigpending\_tsk(struct task\_struct \*t)

{

if (t->signal->group\_stop\_count > 0 ||

(freezing(t)) ||

PENDING(&t->pending, &t->blocked) ||

PENDING(&t->signal->shared\_pending, &t->blocked))

set\_tsk\_thread\_flag(t, TIF\_SIGPENDING);

else

clear\_tsk\_thread\_flag(t, TIF\_SIGPENDING);

}

void recalc\_sigpending(void)

{

recalc\_sigpending\_tsk(current);

}

/\* Given the mask, find the first available signal that should be serviced. \*/

static int

next\_signal(struct sigpending \*pending, sigset\_t \*mask)

{

unsigned long i, \*s, \*m, x;

int sig = 0;

s = pending->signal.sig;

m = mask->sig;

switch (\_NSIG\_WORDS) {

default:

for (i = 0; i < \_NSIG\_WORDS; ++i, ++s, ++m)

if ((x = \*s &~ \*m) != 0) {

sig = ffz(~x) + i\*\_NSIG\_BPW + 1;

break;

}

break;

case 2: if ((x = s[0] &~ m[0]) != 0)

sig = 1;

else if ((x = s[1] &~ m[1]) != 0)

sig = \_NSIG\_BPW + 1;

else

break;

sig += ffz(~x);

break;

case 1: if ((x = \*s &~ \*m) != 0)

sig = ffz(~x) + 1;

break;

}

return sig;

}

static struct sigqueue \*\_\_sigqueue\_alloc(struct task\_struct \*t, gfp\_t flags,

int override\_rlimit)

{

struct sigqueue \*q = NULL;

struct user\_struct \*user;

/\*

\* In order to avoid problems with "switch\_user()", we want to make

\* sure that the compiler doesn't re-load "t->user"

\*/

user = t->user;

barrier();

atomic\_inc(&user->sigpending);

if (override\_rlimit ||

atomic\_read(&user->sigpending) <=

t->signal->rlim[RLIMIT\_SIGPENDING].rlim\_cur)

q = kmem\_cache\_alloc(sigqueue\_cachep, flags);

if (unlikely(q == NULL)) {

atomic\_dec(&user->sigpending);

} else {

INIT\_LIST\_HEAD(&q->list);

q->flags = 0;

q->user = get\_uid(user);

}

return(q);

}

static void \_\_sigqueue\_free(struct sigqueue \*q)

{

if (q->flags & SIGQUEUE\_PREALLOC)

return;

atomic\_dec(&q->user->sigpending);

free\_uid(q->user);

kmem\_cache\_free(sigqueue\_cachep, q);

}

void flush\_sigqueue(struct sigpending \*queue)

{

struct sigqueue \*q;

sigemptyset(&queue->signal);

while (!list\_empty(&queue->list)) {

q = list\_entry(queue->list.next, struct sigqueue , list);

list\_del\_init(&q->list);

\_\_sigqueue\_free(q);

}

}

/\*

\* Flush all pending signals for a task.

\*/

void flush\_signals(struct task\_struct \*t)

{

unsigned long flags;

spin\_lock\_irqsave(&t->sighand->siglock, flags);

clear\_tsk\_thread\_flag(t,TIF\_SIGPENDING);

flush\_sigqueue(&t->pending);

flush\_sigqueue(&t->signal->shared\_pending);

spin\_unlock\_irqrestore(&t->sighand->siglock, flags);

}

/\*

\* Flush all handlers for a task.

\*/

void

flush\_signal\_handlers(struct task\_struct \*t, int force\_default)

{

int i;

struct k\_sigaction \*ka = &t->sighand->action[0];

for (i = \_NSIG ; i != 0 ; i--) {

if (force\_default || ka->sa.sa\_handler != SIG\_IGN)

ka->sa.sa\_handler = SIG\_DFL;

ka->sa.sa\_flags = 0;

sigemptyset(&ka->sa.sa\_mask);

ka++;

}

}

/\* Notify the system that a driver wants to block all signals for this

\* process, and wants to be notified if any signals at all were to be

\* sent/acted upon. If the notifier routine returns non-zero, then the

\* signal will be acted upon after all. If the notifier routine returns 0,

\* then then signal will be blocked. Only one block per process is

\* allowed. priv is a pointer to private data that the notifier routine

\* can use to determine if the signal should be blocked or not. \*/

void

block\_all\_signals(int (\*notifier)(void \*priv), void \*priv, sigset\_t \*mask)

{

unsigned long flags;

spin\_lock\_irqsave(&current->sighand->siglock, flags);

current->notifier\_mask = mask;

current->notifier\_data = priv;

current->notifier = notifier;

spin\_unlock\_irqrestore(&current->sighand->siglock, flags);

}

/\* Notify the system that blocking has ended. \*/

void

unblock\_all\_signals(void)

{

unsigned long flags;

spin\_lock\_irqsave(&current->sighand->siglock, flags);

current->notifier = NULL;

current->notifier\_data = NULL;

recalc\_sigpending();

spin\_unlock\_irqrestore(&current->sighand->siglock, flags);

}

static int collect\_signal(int sig, struct sigpending \*list, siginfo\_t \*info)

{

struct sigqueue \*q, \*first = NULL;

int still\_pending = 0;

if (unlikely(!sigismember(&list->signal, sig)))

return 0;

/\*

\* Collect the siginfo appropriate to this signal. Check if

\* there is another siginfo for the same signal.

\*/

list\_for\_each\_entry(q, &list->list, list) {

if (q->info.si\_signo == sig) {

if (first) {

still\_pending = 1;

break;

}

first = q;

}

}

if (first) {

list\_del\_init(&first->list);

copy\_siginfo(info, &first->info);

\_\_sigqueue\_free(first);

if (!still\_pending)

sigdelset(&list->signal, sig);

} else {

/\* Ok, it wasn't in the queue. This must be

a fast-pathed signal or we must have been

out of queue space. So zero out the info.

\*/

sigdelset(&list->signal, sig);

info->si\_signo = sig;

info->si\_errno = 0;

info->si\_code = 0;

info->si\_pid = 0;

info->si\_uid = 0;

}

return 1;

}

static int \_\_dequeue\_signal(struct sigpending \*pending, sigset\_t \*mask,

siginfo\_t \*info)

{

int sig = next\_signal(pending, mask);

if (sig) {

if (current->notifier) {

if (sigismember(current->notifier\_mask, sig)) {

if (!(current->notifier)(current->notifier\_data)) {

clear\_thread\_flag(TIF\_SIGPENDING);

return 0;

}

}

}

if (!collect\_signal(sig, pending, info))

sig = 0;

}

return sig;

}

/\*

\* Dequeue a signal and return the element to the caller, which is

\* expected to free it.

\*

\* All callers have to hold the siglock.

\*/

int dequeue\_signal(struct task\_struct \*tsk, sigset\_t \*mask, siginfo\_t \*info)

{

int signr = \_\_dequeue\_signal(&tsk->pending, mask, info);

if (!signr)

signr = \_\_dequeue\_signal(&tsk->signal->shared\_pending,

mask, info);

recalc\_sigpending\_tsk(tsk);

if (signr && unlikely(sig\_kernel\_stop(signr))) {

/\*

\* Set a marker that we have dequeued a stop signal. Our

\* caller might release the siglock and then the pending

\* stop signal it is about to process is no longer in the

\* pending bitmasks, but must still be cleared by a SIGCONT

\* (and overruled by a SIGKILL). So those cases clear this

\* shared flag after we've set it. Note that this flag may

\* remain set after the signal we return is ignored or

\* handled. That doesn't matter because its only purpose

\* is to alert stop-signal processing code when another

\* processor has come along and cleared the flag.

\*/

if (!(tsk->signal->flags & SIGNAL\_GROUP\_EXIT))

tsk->signal->flags |= SIGNAL\_STOP\_DEQUEUED;

}

if ( signr &&

((info->si\_code & \_\_SI\_MASK) == \_\_SI\_TIMER) &&

info->si\_sys\_private){

/\*

\* Release the siglock to ensure proper locking order

\* of timer locks outside of siglocks. Note, we leave

\* irqs disabled here, since the posix-timers code is

\* about to disable them again anyway.

\*/

spin\_unlock(&tsk->sighand->siglock);

do\_schedule\_next\_timer(info);

spin\_lock(&tsk->sighand->siglock);

}

return signr;

}

/\*

\* Tell a process that it has a new active signal..

\*

\* NOTE! we rely on the previous spin\_lock to

\* lock interrupts for us! We can only be called with

\* "siglock" held, and the local interrupt must

\* have been disabled when that got acquired!

\*

\* No need to set need\_resched since signal event passing

\* goes through ->blocked

\*/

void signal\_wake\_up(struct task\_struct \*t, int resume)

{

unsigned int mask;

set\_tsk\_thread\_flag(t, TIF\_SIGPENDING);

/\*

\* For SIGKILL, we want to wake it up in the stopped/traced case.

\* We don't check t->state here because there is a race with it

\* executing another processor and just now entering stopped state.

\* By using wake\_up\_state, we ensure the process will wake up and

\* handle its death signal.

\*/

mask = TASK\_INTERRUPTIBLE;

if (resume)

mask |= TASK\_STOPPED | TASK\_TRACED;

if (!wake\_up\_state(t, mask))

kick\_process(t);

}

/\*

\* Remove signals in mask from the pending set and queue.

\* Returns 1 if any signals were found.

\*

\* All callers must be holding the siglock.

\*

\* This version takes a sigset mask and looks at all signals,

\* not just those in the first mask word.

\*/

static int rm\_from\_queue\_full(sigset\_t \*mask, struct sigpending \*s)

{

struct sigqueue \*q, \*n;

sigset\_t m;

sigandsets(&m, mask, &s->signal);

if (sigisemptyset(&m))

return 0;

signandsets(&s->signal, &s->signal, mask);

list\_for\_each\_entry\_safe(q, n, &s->list, list) {

if (sigismember(mask, q->info.si\_signo)) {

list\_del\_init(&q->list);

\_\_sigqueue\_free(q);

}

}

return 1;

}

/\*

\* Remove signals in mask from the pending set and queue.

\* Returns 1 if any signals were found.

\*

\* All callers must be holding the siglock.

\*/

static int rm\_from\_queue(unsigned long mask, struct sigpending \*s)

{

struct sigqueue \*q, \*n;

if (!sigtestsetmask(&s->signal, mask))

return 0;

sigdelsetmask(&s->signal, mask);

list\_for\_each\_entry\_safe(q, n, &s->list, list) {

if (q->info.si\_signo < SIGRTMIN &&

(mask & sigmask(q->info.si\_signo))) {

list\_del\_init(&q->list);

\_\_sigqueue\_free(q);

}

}

return 1;

}

/\*

\* Bad permissions for sending the signal

\*/

static int check\_kill\_permission(int sig, struct siginfo \*info,

struct task\_struct \*t)

{

int error = -EINVAL;

if (!valid\_signal(sig))

return error;

error = -EPERM;

if ((info == SEND\_SIG\_NOINFO || (!is\_si\_special(info) && SI\_FROMUSER(info)))

&& ((sig != SIGCONT) ||

(process\_session(current) != process\_session(t)))

&& (current->euid ^ t->suid) && (current->euid ^ t->uid)

&& (current->uid ^ t->suid) && (current->uid ^ t->uid)

&& !capable(CAP\_KILL))

return error;

error = security\_task\_kill(t, info, sig, 0);

if (!error)

audit\_signal\_info(sig, t); /\* Let audit system see the signal \*/

return error;

}

/\* forward decl \*/

static void do\_notify\_parent\_cldstop(struct task\_struct \*tsk, int why);

/\*

\* Handle magic process-wide effects of stop/continue signals.

\* Unlike the signal actions, these happen immediately at signal-generation

\* time regardless of blocking, ignoring, or handling. This does the

\* actual continuing for SIGCONT, but not the actual stopping for stop

\* signals. The process stop is done as a signal action for SIG\_DFL.

\*/

static void handle\_stop\_signal(int sig, struct task\_struct \*p)

{

struct task\_struct \*t;

if (p->signal->flags & SIGNAL\_GROUP\_EXIT)

/\*

\* The process is in the middle of dying already.

\*/

return;

if (sig\_kernel\_stop(sig)) {

/\*

\* This is a stop signal. Remove SIGCONT from all queues.

\*/

rm\_from\_queue(sigmask(SIGCONT), &p->signal->shared\_pending);

t = p;

do {

rm\_from\_queue(sigmask(SIGCONT), &t->pending);

t = next\_thread(t);

} while (t != p);

} else if (sig == SIGCONT) {

/\*

\* Remove all stop signals from all queues,

\* and wake all threads.

\*/

if (unlikely(p->signal->group\_stop\_count > 0)) {

/\*

\* There was a group stop in progress. We'll

\* pretend it finished before we got here. We are

\* obliged to report it to the parent: if the

\* SIGSTOP happened "after" this SIGCONT, then it

\* would have cleared this pending SIGCONT. If it

\* happened "before" this SIGCONT, then the parent

\* got the SIGCHLD about the stop finishing before

\* the continue happened. We do the notification

\* now, and it's as if the stop had finished and

\* the SIGCHLD was pending on entry to this kill.

\*/

p->signal->group\_stop\_count = 0;

p->signal->flags = SIGNAL\_STOP\_CONTINUED;

spin\_unlock(&p->sighand->siglock);

do\_notify\_parent\_cldstop(p, CLD\_STOPPED);

spin\_lock(&p->sighand->siglock);

}

rm\_from\_queue(SIG\_KERNEL\_STOP\_MASK, &p->signal->shared\_pending);

t = p;

do {

unsigned int state;

rm\_from\_queue(SIG\_KERNEL\_STOP\_MASK, &t->pending);

/\*

\* If there is a handler for SIGCONT, we must make

\* sure that no thread returns to user mode before

\* we post the signal, in case it was the only

\* thread eligible to run the signal handler--then

\* it must not do anything between resuming and

\* running the handler. With the TIF\_SIGPENDING

\* flag set, the thread will pause and acquire the

\* siglock that we hold now and until we've queued

\* the pending signal.

\*

\* Wake up the stopped thread \_after\_ setting

\* TIF\_SIGPENDING

\*/

state = TASK\_STOPPED;

if (sig\_user\_defined(t, SIGCONT) && !sigismember(&t->blocked, SIGCONT)) {

set\_tsk\_thread\_flag(t, TIF\_SIGPENDING);

state |= TASK\_INTERRUPTIBLE;

}

wake\_up\_state(t, state);

t = next\_thread(t);

} while (t != p);

if (p->signal->flags & SIGNAL\_STOP\_STOPPED) {

/\*

\* We were in fact stopped, and are now continued.

\* Notify the parent with CLD\_CONTINUED.

\*/

p->signal->flags = SIGNAL\_STOP\_CONTINUED;

p->signal->group\_exit\_code = 0;

spin\_unlock(&p->sighand->siglock);

do\_notify\_parent\_cldstop(p, CLD\_CONTINUED);

spin\_lock(&p->sighand->siglock);

} else {

/\* We are not stopped, but there could be a stop

\* signal in the middle of being processed after

\* being removed from the queue. Clear that too. \*/

p->signal->flags = 0;

}

} else if (sig == SIGKILL) {

/\*

\* Make sure that any pending stop signal already dequeued

\* is undone by the wakeup for SIGKILL.

\*/

p->signal->flags = 0;

}

}

static int send\_signal(int sig, struct siginfo \*info, struct task\_struct \*t,

struct sigpending \*signals)

{

struct sigqueue \* q = NULL;

int ret = 0;

/\*

\* fast-pathed signals for kernel-internal things like SIGSTOP

\* or SIGKILL.

\*/

if (info == SEND\_SIG\_FORCED)

goto out\_set;

/\* Real-time signals must be queued if sent by sigqueue, or

some other real-time mechanism. It is implementation

defined whether kill() does so. We attempt to do so, on

the principle of least surprise, but since kill is not

allowed to fail with EAGAIN when low on memory we just

make sure at least one signal gets delivered and don't

pass on the info struct. \*/

q = \_\_sigqueue\_alloc(t, GFP\_ATOMIC, (sig < SIGRTMIN &&

(is\_si\_special(info) ||

info->si\_code >= 0)));

if (q) {

list\_add\_tail(&q->list, &signals->list);

switch ((unsigned long) info) {

case (unsigned long) SEND\_SIG\_NOINFO:

q->info.si\_signo = sig;

q->info.si\_errno = 0;

q->info.si\_code = SI\_USER;

q->info.si\_pid = current->pid;

q->info.si\_uid = current->uid;

break;

case (unsigned long) SEND\_SIG\_PRIV:

q->info.si\_signo = sig;

q->info.si\_errno = 0;

q->info.si\_code = SI\_KERNEL;

q->info.si\_pid = 0;

q->info.si\_uid = 0;

break;

default:

copy\_siginfo(&q->info, info);

break;

}

} else if (!is\_si\_special(info)) {

if (sig >= SIGRTMIN && info->si\_code != SI\_USER)

/\*

\* Queue overflow, abort. We may abort if the signal was rt

\* and sent by user using something other than kill().

\*/

return -EAGAIN;

}

out\_set: sigaddset(&signals->signal, sig);

return ret;

}

#define LEGACY\_QUEUE(sigptr, sig) \

(((sig) < SIGRTMIN) && sigismember(&(sigptr)->signal, (sig)))

static int

specific\_send\_sig\_info(int sig, struct siginfo \*info, struct task\_struct \*t)

{

int ret = 0;

BUG\_ON(!irqs\_disabled());

assert\_spin\_locked(&t->sighand->siglock);

/\* Short-circuit ignored signals. \*/

if (sig\_ignored(t, sig))

goto out;

/\* Support queueing exactly one non-rt signal, so that we

can get more detailed information about the cause of

the signal. \*/

if (LEGACY\_QUEUE(&t->pending, sig))

goto out;

ret = send\_signal(sig, info, t, &t->pending);

if (!ret && !sigismember(&t->blocked, sig))

signal\_wake\_up(t, sig == SIGKILL);

out:

return ret;

}

#include "stdafx.h"

#include "finding.h"

#include "MainFrm.h"

#include "findingDoc.h"

#include "findingView.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

BEGIN\_MESSAGE\_MAP(CFindingApp, CWinApp)

//{{AFX\_MSG\_MAP(CFindingApp)

ON\_COMMAND(ID\_APP\_ABOUT, OnAppAbout)

// 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 file based document commands

ON\_COMMAND(ID\_FILE\_NEW, CWinApp::OnFileNew)

ON\_COMMAND(ID\_FILE\_OPEN, CWinApp::OnFileOpen)

// Standard print setup command

ON\_COMMAND(ID\_FILE\_PRINT\_SETUP, CWinApp::OnFilePrintSetup)

END\_MESSAGE\_MAP()

CFindingApp::CFindingApp()

{

// TODO: add construction code here,

// Place all significant initialization in InitInstance

}

CFindingApp theApp;

BOOL CFindingApp::InitInstance()

{

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

// Change the registry key under which our settings are stored.

// TODO: You should modify this string to be something appropriate

// such as the name of your company or organization.

SetRegistryKey(\_T("Local AppWizard-Generated Applications"));

LoadStdProfileSettings(); // Load standard INI file options (including MRU)

// Register the application's document templates. Document templates

// serve as the connection between documents, frame windows and views.

CSingleDocTemplate\* pDocTemplate;

pDocTemplate = new CSingleDocTemplate(

IDR\_MAINFRAME,

RUNTIME\_CLASS(CFindingDoc),

RUNTIME\_CLASS(CMainFrame), // main SDI frame window

RUNTIME\_CLASS(CFindingView));

AddDocTemplate(pDocTemplate);

// Parse command line for standard shell commands, DDE, file open

CCommandLineInfo cmdInfo;

ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line

if (!ProcessShellCommand(cmdInfo))

return FALSE;

// The one and only window has been initialized, so show and update it.

m\_pMainWnd->ShowWindow(SW\_SHOW);

m\_pMainWnd->UpdateWindow();

return TRUE;

}

class CAboutDlg : public CDialog

{

public:

CAboutDlg();

// Dialog Data

//{{AFX\_DATA(CAboutDlg)

enum { IDD = IDD\_ABOUTBOX };

//}}AFX\_DATA

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CAboutDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

// Implementation

protected:

//{{AFX\_MSG(CAboutDlg)

// No message handlers

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

{

//{{AFX\_DATA\_INIT(CAboutDlg)

//}}AFX\_DATA\_INIT

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CAboutDlg)

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialog)

//{{AFX\_MSG\_MAP(CAboutDlg)

// No message handlers

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

// App command to run the dialog

void CFindingApp::OnAppAbout()

{

CAboutDlg aboutDlg;

aboutDlg.DoModal();

}

#include "stdafx.h"

#include "keyboard.h"

#include "MainFrm.h"

#include "keyboardDoc.h"

#include "keyboardView.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

BEGIN\_MESSAGE\_MAP(CKeyboardApp, CWinApp)

//{{AFX\_MSG\_MAP(CKeyboardApp)

ON\_COMMAND(ID\_APP\_ABOUT, OnAppAbout)

// 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 file based document commands

ON\_COMMAND(ID\_FILE\_NEW, CWinApp::OnFileNew)

ON\_COMMAND(ID\_FILE\_OPEN, CWinApp::OnFileOpen)

// Standard print setup command

ON\_COMMAND(ID\_FILE\_PRINT\_SETUP, CWinApp::OnFilePrintSetup)

END\_MESSAGE\_MAP()

CKeyboardApp::CKeyboardApp()

{

// TODO: add construction code here,

// Place all significant initialization in InitInstance

}

CKeyboardApp theApp;

BOOL CKeyboardApp::InitInstance()

{

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

SetRegistryKey(\_T("Local AppWizard-Generated Applications"));

LoadStdProfileSettings(); // Load standard INI file options (including MRU)

// Register the application's document templates. Document templates

// serve as the connection between documents, frame windows and views.

CSingleDocTemplate\* pDocTemplate;

pDocTemplate = new CSingleDocTemplate(

IDR\_MAINFRAME,

RUNTIME\_CLASS(CKeyboardDoc),

RUNTIME\_CLASS(CMainFrame), // main SDI frame window

RUNTIME\_CLASS(CKeyboardView));

AddDocTemplate(pDocTemplate);

// Parse command line for standard shell commands, DDE, file open

CCommandLineInfo cmdInfo;

ParseCommandLine(cmdInfo);

if (!ProcessShellCommand(cmdInfo))

return FALSE;

m\_pMainWnd->ShowWindow(SW\_SHOW);

m\_pMainWnd->UpdateWindow();

return TRUE;

}

class CAboutDlg : public CDialog

{

public:

CAboutDlg();

// Dialog Data

//{{AFX\_DATA(CAboutDlg)

enum { IDD = IDD\_ABOUTBOX };

//}}AFX\_DATA

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CAboutDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

// Implementation

protected:

//{{AFX\_MSG(CAboutDlg)

// No message handlers

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

{

//{{AFX\_DATA\_INIT(CAboutDlg)

//}}AFX\_DATA\_INIT

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CAboutDlg)

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialog)

//{{AFX\_MSG\_MAP(CAboutDlg)

// No message handlers

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

// App command to run the dialog

void CKeyboardApp::OnAppAbout()

{

CAboutDlg aboutDlg;

aboutDlg.DoModal();

}

BOOL CFileManipulateApp::InitInstance()

{

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

CFileManipulateDlg dlg;

m\_pMainWnd = &dlg;

int nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: Place code here to handle when the dialog is

// dismissed with OK

}

else if (nResponse == IDCANCEL)

{

// TODO: Place code here to handle when the dialog is

// dismissed with Cancel

}

// Since the dialog has been closed, return FALSE so that we exit the

// application, rather than start the application's message pump.

return FALSE;

}

class CAboutDlg : public CDialog

{

public:

CAboutDlg();

enum { IDD = IDD\_ABOUTBOX };

//}}AFX\_DATA

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CAboutDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

protected:

//{{AFX\_MSG(CAboutDlg)

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

{

//{{AFX\_DATA\_INIT(CAboutDlg)

//}}AFX\_DATA\_INIT

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CAboutDlg)

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialog)

//{{AFX\_MSG\_MAP(CAboutDlg)

// No message handlers

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

CFileManipulateDlg::CFileManipulateDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CFileManipulateDlg::IDD, pParent)

{

m\_szDstFile = \_T("");

m\_szSrcFile = \_T("");

//}}AFX\_DATA\_INIT

// Note that LoadIcon does not require a subsequent DestroyIcon in Win32

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

void CFileManipulateDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CFileManipulateDlg)

DDX\_Text(pDX, IDC\_DSTFILE, m\_szDstFile);

DDX\_Text(pDX, IDC\_SRCFILE, m\_szSrcFile);

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(CFileManipulateDlg, CDialog)

//{{AFX\_MSG\_MAP(CFileManipulateDlg)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_OPENSRC, OnOpenSrc)

ON\_BN\_CLICKED(IDC\_OPENDST, OnOpenDst)

ON\_BN\_CLICKED(IDC\_DEL, OnDel)

ON\_BN\_CLICKED(IDC\_COPY, OnCopy)

ON\_BN\_CLICKED(IDC\_MOVE, OnMove)

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

BOOL CFileManipulateDlg::OnInitDialog()

{

CDialog::OnInitDialog();

// Add "About..." menu item to system menu.

// IDM\_ABOUTBOX must be in the system command range.

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

CString strAboutMenu;

strAboutMenu.LoadString(IDS\_ABOUTBOX);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

SetIcon(m\_hIcon, TRUE); // Set big icon

SetIcon(m\_hIcon, FALSE); // Set small icon

TODO: Add extra initialization here

return TRUE; // return TRUE unless you set the focus to a control

}

void CFileManipulateDlg::OnSysCommand(UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

else

{

CDialog::OnSysCommand(nID, lParam);

}

}

void CFileManipulateDlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // device context for painting

SendMessage(WM\_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

// Center icon in client rectangle

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CDialog::OnPaint();

}

}

HCURSOR CFileManipulateDlg::OnQueryDragIcon()

{

return (HCURSOR) m\_hIcon;

}

#endif