PROGRAM LISTING

```
// FILE StdAfx.h

#include <afxwin.h>        // MFC core and standard components
#include <afxext.h>        // MFC extensions (including VB)
#include <afxole.h>        // MFC OLE 2.0 support
#include <afxodlgs.h>      // MFC OLE 2.0 dialog support

// FILE stdafx.cpp : source file that includes just the standard includes

//  stdafx.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"


// FILE PlcApp.h.  See plcapp.cpp for the implementation of this class

class CPLCApp : public CWinApp
{
public:
        virtual BOOL InitInstance();
        CPLCApp(){}
        CMultiDocTemplate* m_pInstrDocTemplate;

// Implementation
protected:
        //{{AFX_MSG(CPLCApp)
        afx_msg void OnAppAbout();
                // 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()
};


// FILE plcapp.cpp : Defines the class behaviors for the application.

#include "stdafx.h"
#include "plcapp.h"
#include "mainfrm.h"
#include "plcobj.h"
#include "plcdoc.h"
#include "plcview.h"
#include "instrvw.h"
#include "splitfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

BEGIN_MESSAGE_MAP(CPLCApp, CWinApp)
```

```
	//{{AFX_MSG_MAP(CPLCApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
	//}}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()
```

// FILE PLCApp.CPP

```
// The one and only CPLCApp object
CPLCApp theApp;

// CPLCApp initialization
//To intialize each new instant of application running under windows
BOOL CPLCApp::InitInstance()
{
	// Initialize OLE 2.0 libraries
	if (!AfxOleInit())
	{
		AfxMessageBox(IDP_OLE_INIT_FAILED);
		return FALSE;
	}
	// Standard initialization
	Enable3dControls();	// enable 3d controls in dialogs
	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_PLCTYPE, RUNTIME_CLASS(CPLCDoc),
		RUNTIME_CLASS(CSplitFrame), RUNTIME_CLASS(CPLCView));
	pDocTemplate->SetContainerInfo(IDR_PLCTYPE_CNTR_IP);
	AddDocTemplate(pDocTemplate);

	// create main MDI Frame window
	CMainFrame* pMainFrame = new CMainFrame;
	if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
	//load the main frame and attach to which to attach to CFrameWnd
		return FALSE; // frame window creation failed.

	m_pMainWnd = pMainFrame;

	//To prodive connection b/w Doc,Frame,View
	// for new application Instruct doc (with menu) added
	m_pInstrDocTemplate = new CMultiDocTemplate(
		IDR_INSTR_TYPE,
		RUNTIME_CLASS(CInstructDoc),
		RUNTIME_CLASS(CInstructFrame),
		RUNTIME_CLASS(CInstructView));
	AddDocTemplate(m_pInstrDocTemplate);
```

```
	// enable file manager drag/drop and DDE Execute open
	m_pMainWnd->DragAcceptFiles();
	EnableShellOpen();
	RegisterShellFileTypes();

	// simple command line parsing
	if (m_lpCmdLine[0] == '\0')
	{
		// create a new (empty) document
		OnFileNew();
	}
	else
	{
		// open an existing document
		OpenDocumentFile(m_lpCmdLine);
	}
	OnIdle(0);  // updates buttons before showing the window
	// 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

// Implementation
protected:
	virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
	//{{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 CPLCApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

```
// FILE mainfrm.h : interface of the CMainFrame class
```

```
class CMainFrame : public CMDIFrameWnd
{
	DECLARE_DYNAMIC(CMainFrame)
public:
	CMainFrame();
	void SetStatusMessage( int nStringID  )
	{
		CString strMsg;
		strMsg.LoadString( nStringID );
		m_wndStatusBar.SetPaneText( 0, strMsg );
	}
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
protected:
	CStatusBar  m_wndStatusBar; // control bar embedded members
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	//{{AFX_MSG(CMainFrame)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
		// 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()
};
```

```
// FILE mainfrm.cpp : implementation of the CMainFrame class
```

```
#include "stdafx.h"
#include "plcapp.h"
#include "mainfrm.h"
```

```
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CMainFrame
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)  //relate to runtime
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// arrays of IDs used to initialize control bars
// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
	// same order as in the bitmap 'toolbar.bmp'
	ID_FILE_NEW,
	ID_FILE_OPEN,
	ID_FILE_SAVE,

	ID_SEPARATOR,
	ID_EDIT_CUT,
	ID_EDIT_COPY,
	ID_EDIT_PASTE,

	ID_SEPARATOR,
	ID_DRAW_SELECT,
	ID_DRAW_LINE,
	ID_DRAW_SHUNT,
	ID_DRAW_UPWARD_SHUNT,

	ID_SEPARATOR,
	ID_DRAW_MOVE,
	ID_DRAW_GREATERTHAN,
	ID_DRAW_LESSTHAN,
	ID_DRAW_EQUALTO,
	ID_DRAW_ADDITION,
	ID_DRAW_SUBTRACTION,

	ID_SEPARATOR,
	ID_DRAW_SERIAL_OPEN_SWITCH,
	ID_DRAW_SERIAL_CLOSED_SWITCH,
	ID_DRAW_OUTPUT,
	ID_DRAW_RESET_SWITCH,
	ID_DRAW_SET_SWITCH,

	ID_SEPARATOR,
	ID_DRAW_START,
	ID_DRAW_STOP,
	ID_DRAW_COUNTER,
	ID_DRAW_TIMER,
```

```
	ID_SEPARATOR,
	ID_FILE_PRINT,
	ID_APP_ABOUT,
};

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

CMainFrame::CMainFrame()// CMainFrame construction/destruction
{
}

CMainFrame::~CMainFrame()
{
}

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

	if (!m_wndToolBar.Create(this, WS_CHILD | WS_VISIBLE |
		CBRS_TOP | CBRS_FLYBY | CBRS_TOOLTIPS ) ||
		!m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
		!m_wndToolBar.SetButtons(buttons,
		  sizeof(buttons)/sizeof(UINT)))
	{
		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
	}

	return 0;
}

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

```
void CMainFrame::Dump(CDumpContext& dc) const
{
	CMDIFrameWnd::Dump(dc);
}
#endif //_DEBUG

// plcapp.h :
#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"	// main symbols
#include <afxtempl.h>	// plcapp uses collection class templates
```

```
// FILE PLCdoc.h : interface of the CPLCDoc class
```

```
#include "plcobj.h"
#include "instruct.h"

class CPLCView;
class CCursorObj : public CObject
{
	DECLARE_SERIAL(CCursorObj)
	public:
		CCursorObj();
		~CCursorObj(){}

		virtual void Serialize(CArchive& ar);
		CDrawRect* m_pCursor;
		int m_nCurrentLineY; // tells us the Y value of the cursor
		int m_nCurrentLineX; // tells us the X value of the cursor
		int m_nCurrentLineNum;
		CDrawRect* m_pCurrentLine;
		int m_nNextLineNumber;
		int m_nLastRungY; // used to find the Y to create new rung
		int m_nAbsoluteLastY;
};

class CPLCDoc : public COleDocument
{
protected: // create from serialization only
	CPLCDoc();
	DECLARE_DYNCREATE(CPLCDoc)

public:
	CDrawObjList* GetObjects() { return &m_objects; }
	void MoveRightLine( CPLCView* );
	const CSize& GetSize() const { return m_size; }
	void ComputePageSize( int x = 0, int y = 0 );
	int GetMapMode() const { return m_nMapMode; }
	COLORREF GetPaperColor() const { return m_paperColor; }
	CDrawObj* NameExists( LPCSTR pszValue, DrawShape drawShape );
	void DeleteVertical( CPLCView* );
```

```
        CDrawObj* ObjectAt(const CPoint& point);
        void Draw(CDC* pDC, CPLCView* pView);
        void Add(CDrawObj* pObj);
        void Remove(CDrawObj* pObj, BOOL bForceRemove = FALSE );
        int GetLineY( int nLineNum );
        CDrawRect* GetLine( int nLineNum );
        BOOL EndAdded(){ return m_bEndAdded; }
        int GetEndRungY(){ return m_nEndY; }
        void SetEndRungY( int nY){ m_nEndY = nY; }
        CCursorObj m_cursor;
        void ChangeMaxYBy( int nSize ){ m_nMaxY += nSize; }

        CDrawObj* GetLastObject()
        {
                POSITION pos = m_objects.GetTailPosition();
                if(pos)
                        return m_objects.GetPrev(pos);
                else
                        return NULL;
        }

        BOOL m_bStartAdded;
        int m_nNextObjectNumber;
        int m_nMaxY;
        int LINE_END_X;
        CDrawRect* m_startLine;

// Implementation
public:
        virtual ~CPLCDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
        CLadder* GetLadder(){ return &m_Ladder; }

protected:
        virtual BOOL OnNewDocument();

        CDrawObjList m_objects;
        CLadder m_Ladder;
        CSize m_size;
        int m_nMapMode;
        COLORREF m_paperColor;
protected:
        //{{AFX_MSG(CPLCDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
                //   DO NOT EDIT what you see in these blocks of generated code !
        afx_msg void OnViewPaperColor();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

        private:
```

```
		void FixupPointers();
		CDrawObj* FindDrawObj( int nObj );
		void FixRung( CInstructRung* pRung );
		void FixChildRung( CInstructRung* pRung );

		int m_nEndY;
		BOOL m_bEndAdded;

};
```

// plcdoc.cpp : implementation of the CPLCDoc class

```
#include "stdafx.h"
#include "plcapp.h"
#include "plcdoc.h"
#include "plcview.h"
#include "plcobj.h"
#include "cntritem.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CPLCDoc, COleDocument)
	//{{AFX_MSG_MAP(CPLCDoc)
BEGIN_MESSAGE_MAP(CPLCDoc, COleDocument)
	ON_COMMAND(ID_VIEW_PAPERCOLOR, OnViewPaperColor)
	//}}AFX_MSG_MAP
	// Enable default OLE container implementation
	ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE,
		COleDocument::OnUpdatePasteMenu)
	ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE_LINK,
		COleDocument::OnUpdatePasteLinkMenu)
	ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_LINKS,
		COleDocument::OnUpdateEditLinksMenu)
	ON_COMMAND(ID_OLE_EDIT_LINKS, COleDocument::OnEditLinks)
	ON_UPDATE_COMMAND_UI(ID_OLE_VERB_FIRST,
		COleDocument::OnUpdateObjectVerbMenu)
	ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_CONVERT,
		COleDocument::OnUpdateObjectVerbMenu)
	ON_COMMAND(ID_OLE_EDIT_CONVERT,
		COleDocument::OnEditConvert)
END_MESSAGE_MAP()

IMPLEMENT_SERIAL(CCursorObj, CObject,0)
CCursorObj::CCursorObj()
{
	m_nCurrentLineY = START_LINE_BEGIN_Y;
	m_nCurrentLineX = CURSOR_X;
	m_nNextLineNumber = 0;
	m_nCurrentLineNum = 0;
	m_pCursor = NULL;
	m_nLastRungY = START_LINE_BEGIN_Y;
```

```
	m_nAbsoluteLastY = 0;
}

void CCursorObj::Serialize(CArchive& ar)
{
	CObject::Serialize(ar);

	if (ar.IsStoring())
	{
		ar << (WORD)m_pCursor->m_nObjectNumber;
		ar << (WORD)m_nCurrentLineY; // tells us the Y value of the cursor
		ar << (WORD)m_nCurrentLineX; // tells us the X value of the cursor
		ar << (WORD)m_nCurrentLineNum;
		ar << (WORD)m_nNextLineNumber;
		ar << (DWORD)m_pCurrentLine->m_nLineNumber;
	}
	else
	{
		WORD wTemp;
		ar >> wTemp; m_pCursor = (CDrawRect*)wTemp;
		ar >> wTemp; m_nCurrentLineY = wTemp;
		// tells us the X value of the cursor
		ar >> wTemp; m_nCurrentLineX = wTemp;

		ar >> wTemp; m_nCurrentLineNum = wTemp;
		ar >> wTemp; m_nNextLineNumber = wTemp;
		DWORD dwTemp;
		ar >> dwTemp; m_pCurrentLine = (CDrawRect*)dwTemp;
	}
}

CPLCDoc::CPLCDoc()// CPLCDoc construction/destruction
{
	m_nMapMode = MM_ANISOTROPIC;
	m_paperColor = RGB(255, 255, 255);
	ComputePageSize();
	m_bStartAdded = FALSE;
	m_bEndAdded = FALSE;
	m_nNextObjectNumber = 0;
	m_startLine = NULL;

	m_nMaxY = START_LINE_BEGIN_Y;
	LINE_END_X = 120; //700;
}

CPLCDoc::~CPLCDoc()
{
	POSITION pos = m_objects.GetHeadPosition();
	while (pos != NULL)
		delete m_objects.GetNext(pos);
}

BOOL CPLCDoc::OnNewDocument()
{
```

```
	if (!COleDocument::OnNewDocument())
		return FALSE;
	// (SDI documents will reuse this document)
	return TRUE;
}

// CPLCDoc serialization
CDrawObj* CPLCDoc::FindDrawObj( int nObj )
{
	POSITION pos = m_objects.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_objects.GetNext(pos);
		if( pObj->m_nObjectNumber == nObj )
			return pObj;
	}
	ASSERT(FALSE);
	return NULL;
}

void CPLCDoc::MoveRightLine( CPLCView* pView )
{
	POSITION pos = m_objects.GetHeadPosition();
	POSITION oldPos;

	// start has two lines. only extend the second line to right
	BOOL bStartSeen = FALSE;

	while (pos != NULL)
	{
		oldPos = pos;
		CDrawObj* pObj = m_objects.GetNext(pos);
		if( pObj->m_drawShape == line )
		{
			CRect rect = pObj->m_position;
			pView->DocToClient( rect );

			CDrawRect::LineType LType = ((CDrawRect*)pObj)->m_LineType;

			if( LType == CDrawRect::right_vertical )
			{
				// remove vertical line
				delete pObj;
				m_objects.RemoveAt(oldPos);
			}
			else
			if( LType == CDrawRect::start || LType == CDrawRect::end ||
				LType == CDrawRect::rung ) // if horizontal
			{
				if( LType == CDrawRect::start && !bStartSeen )
				{
					bStartSeen = TRUE;
					// extend horizontal line
					rect.right += DEFAULT_X_INCREMENT/2;
```

```
                                        pView->ClientToDoc( rect );
                                        pObj->m_position = rect;
                                }
                                else
                                {
                                        // extend horizontal line
                                        if( LType == CDrawRect::start )
                                        {
                                                rect.left += DEFAULT_X_INCREMENT/2;
                                                rect.right += DEFAULT_X_INCREMENT;
                                        }
                                        else
                                                rect.right += DEFAULT_X_INCREMENT ;
                                        pView->ClientToDoc( rect );
                                        pObj->m_position = rect;
                                }
                        }
                }
                else
                if( pObj->m_drawShape == plc_start )
                {
                        // move the start switch
                        CRect rect = pObj->m_position;
                        pView->DocToClient( rect );
                        rect.left += DEFAULT_X_INCREMENT/2;
                        rect.right += DEFAULT_X_INCREMENT/2;
                        pView->ClientToDoc( rect );
                        if( rect.top > rect.bottom )
                        {
                                int a = rect.top;
                                rect.top = rect.bottom;
                                rect.bottom = a;
                        }
                        pObj->m_position = rect;
                        pView->Invalidate( );
                }
        }
}

void CPLCDoc::DeleteVertical( CPLCView* pView )
{
        POSITION pos = m_objects.GetHeadPosition();
        POSITION oldPos;
        while (pos != NULL)
        {
                oldPos = pos;
                CDrawObj* pObj = m_objects.GetNext(pos);
                if( pObj->m_drawShape == line &&
                        ((CDrawRect*)pObj)->m_LineType == CDrawRect::right_vertical )
                {
                        delete pObj; // remove vertical line
                        m_objects.RemoveAt(oldPos);
                }
        }
```

```
}

void CPLCDoc::FixRung( CInstructRung* pRung )
{
	pRung->m_pMainRungLine = GetLine( (int)pRung->m_pMainRungLine );
	if( pRung->m_pParallelRungLeftLine )
		pRung->m_pParallelRungLeftLine =
			GetLine( (int)pRung->m_pParallelRungLeftLine );

	if( pRung->m_pParallelRungRightLine )
		pRung->m_pParallelRungRightLine =
			GetLine( (int)pRung->m_pParallelRungRightLine );

	if( pRung->m_pParentRung )
		pRung->m_pParentRung = m_Ladder.
			GetRung( (int)pRung->m_pParentRung );

	CInstruction* pInstruct;
	int nSize = pRung->m_arInstructions.GetSize();
	for( int i = 0; i < nSize; i++ )
	{
		pInstruct = (CInstruction*)pRung->m_arInstructions[i];
		if( pInstruct->m_Type == CInstruction::Instruction )
		{
			pInstruct->m_pDrawObj =
				FindDrawObj( (int)pInstruct->m_pDrawObj );
			pInstruct->m_pOpCode = pInstruct->GetOpCode();
		}
		else
		if( pInstruct->m_Type == CInstruction::ParallelBegin )
			FixRung( pInstruct->m_pChildRung );
	}
}

void CPLCDoc::FixChildRung( CInstructRung* pRung )
{
	CInstruction* pInstruct;
	int nSize = pRung->m_arInstructions.GetSize();
	for( int i = 0; i < nSize; i++ )
	{
		pInstruct = (CInstruction*)pRung->m_arInstructions[i];
		if( pInstruct->m_Type == CInstruction::ParallelBegin )
		{
			pInstruct->m_pChildRung = m_Ladder.
				GetRung( (int)pInstruct->m_pChildRung );
			FixChildRung( pInstruct->m_pChildRung );
		}
	}
}

void CPLCDoc::FixupPointers()
{
	CInstructRung* pRung;
```

```
	POSITION posRung = m_Ladder.m_arInstructionRungs.GetHeadPosition();
	while( posRung )
	{
		pRung = m_Ladder.m_arInstructionRungs.GetNext(posRung);

		FixChildRung( pRung );
	}

	posRung = m_Ladder.m_arInstructionRungs.GetHeadPosition();
	while( posRung )
	{
		pRung = m_Ladder.m_arInstructionRungs.GetNext(posRung);

		FixRung( pRung );
	}
}

void CPLCDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		ar << m_paperColor;
		ar << (WORD)m_nNextObjectNumber;
		ar << (WORD)m_nMaxY;
		ar << (WORD)LINE_END_X;

		m_objects.Serialize(ar);
		m_Ladder.Serialize(ar);
		m_cursor.Serialize(ar);

		if( m_startLine )
			ar << (WORD)m_startLine->m_nLineNumber;
		else
			ar << (WORD)0;
	}
	else
	{
		ar >> m_paperColor;
		WORD wTemp;
		ar >> wTemp;
		m_nNextObjectNumber = wTemp;
		ar >> wTemp; m_nMaxY = wTemp;
		ar >> wTemp; LINE_END_X = wTemp;

		m_objects.Serialize(ar);
		m_Ladder.Serialize(ar);

		// check if the start and end object is available
		POSITION pos = m_objects.GetHeadPosition();
		while( pos )
		{
			CDrawObj* pDrawObj = m_objects.GetNext( pos );
			if( pDrawObj->m_drawShape == plc_start )
			{
```

```
                        m_bStartAdded = TRUE;
                }
                else
                if( pDrawObj->m_drawShape == plc_end )
                {
                        m_bEndAdded = TRUE;
                }

                if( pDrawObj->IsKindOf( RUNTIME_CLASS(CDrawBitmap )))
                {
                        if( (int)((CDrawBitmap*)pDrawObj)->m_pLine )
                                ((CDrawBitmap*)pDrawObj)->m_pLine =
                                GetLine( (int)((CDrawBitmap*)pDrawObj)->m_pLine );
                }
                else
                if( pDrawObj->IsKindOf( RUNTIME_CLASS(CDrawRect)))
                {
                        ((CDrawRect*)pDrawObj)->m_pRung = m_Ladder.
                                GetRung( (int)((CDrawRect*)pDrawObj)->m_pRung );
                }

                if( m_bEndAdded && m_bStartAdded )
                        break;
        }

        FixupPointers();

        m_cursor.Serialize(ar);
        m_cursor.m_pCurrentLine =
                GetLine( (int)m_cursor.m_pCurrentLine );

        m_cursor.m_pCursor =
                (CDrawRect*)FindDrawObj( (int)m_cursor.m_pCursor );

        ar >> wTemp;
        if( wTemp )
        m_startLine = GetLine( (int)wTemp );
    }

    // By calling the base class COleDocument, we enable serialization
    // of the container document's COleClientItem objects automatically.
    COleDocument::Serialize(ar);
}

// CPLCDoc implementation
// Call this fuction to draw the image to View(ie on screen),Printer.

void CPLCDoc::Draw(CDC* pDC, CPLCView* pView)
{
    POSITION pos = m_objects.GetHeadPosition();
    while (pos != NULL)
    {
        CDrawObj* pObj = m_objects.GetNext(pos);
        pObj->Draw(pDC);
```

```
		if (pView->m_bActive && !pDC->IsPrinting()
			&& pView->IsSelected(pObj))
			pObj->DrawTracker(pDC, CDrawObj::selected);
	}
}

//	Adds a new element or list of element to the tail of this list.
//The list may be empty before the operation.
void CPLCDoc::Add(CDrawObj* pObj)
{
	m_objects.AddTail(pObj);

	if( pObj->m_drawShape == plc_start )
		m_bStartAdded = TRUE;
	else
	if( pObj->m_drawShape == plc_end )
	{
		m_bEndAdded = TRUE;
		CRect rect = ((CDrawBitmap*)pObj)->m_pLine->m_position;

		POSITION pos = GetFirstViewPosition();
		CPLCView* pView = NULL;
		if( pos != NULL )
			pView = (CPLCView*)GetNextView(pos);

		if( pView )
			pView->DocToClient( rect );
		else
			ASSERT(FALSE);

		m_nEndY = rect.bottom;
	}

	pObj->m_pDocument = this;
	pObj->m_nObjectNumber = m_nNextObjectNumber++;

	//By calling this function consistently, you ensure that the
	//framework prompts the user to save change before closing a document.
	SetModifiedFlag();
}

//Remove an element from this list.
void CPLCDoc::Remove(CDrawObj* pObj, BOOL bForceRemove )
{
	// don't remove if the object is not deletable
	if( !pObj->m_bDeletable && !bForceRemove )
		return;

	// Find and remove from document
	POSITION pos = m_objects.Find(pObj);
	if (pos != NULL)
		m_objects.RemoveAt(pos);
	// set document modified flag
	SetModifiedFlag();
```

```
	// call remove for each view so
	// that the view can remove from m_selection
	pos = GetFirstViewPosition();
	while (pos != NULL)
		((CPLCView*)GetNextView(pos))->Remove(pobj);
}

CDrawRect* CPLCDoc::GetLine( int nLineNum )
{
	POSITION pos = m_objects.GetTailPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_objects.GetPrev(pos);
		if( pObj->m_drawShape == line && ((CDrawRect*)pObj)->
			m_nLineNumber == nLineNum )
		{
			return (CDrawRect*)pObj;
		}
	}
	return NULL;
}

int CPLCDoc::GetLineY( int nLineNum )
{
	POSITION pos = m_objects.GetTailPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_objects.GetPrev(pos);
		if( pObj->m_drawShape == line &&
			((CDrawRect*)pObj)->m_nLineNumber == nLineNum )
		{
			CRect rect = pObj->m_position;
			POSITION pos = GetFirstViewPosition();
			if(pos != NULL)
			{
				CPLCView* pView = (CPLCView*)GetNextView(pos);
				pView->DocToClient(rect);
				return rect.top;
			}
			return 0;
		}
	}
	return 0;
}

CDrawObj* CPLCDoc::NameExists( LPCSTR pszValue, DrawShape drawShape )
{
	POSITION pos = m_objects.GetTailPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_objects.GetPrev(pos);

		if( pObj->m_strName == pszValue )
```

```
		{
			if( pObj->m_drawShape == drawShape )
				return pObj;
			else
			//open and closed switches cannot have same names
			 if( ( drawShape == serial_open ||  drawShape == parallel_open || drawShape ==
				serial_closed || drawShape == parallel_closed ) && ( pObj-
				>m_drawShape == serial_open || pObj->m_drawShape ==
				parallel_open || pObj->m_drawShape == serial_closed ||
				pObj->m_drawShape == parallel_closed ) )
				return pObj;
		}
	}
	return NULL;
}

CDrawObj* CPLCDoc::ObjectAt(const CPoint& point) // point is in logical coordinates
{
	CRect rect(point, CSize(1, 1));
	POSITION pos = m_objects.GetTailPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_objects.GetPrev(pos);

		// if the line object is encountered, increase its x dim by one,
		if( pObj->m_drawShape == line )
		{
			CRect lineRect = pObj->m_position;
			if( lineRect.left == lineRect.right )
			{
				lineRect.left -= 2;
				lineRect.right+= 2;
			}
			else
			if( lineRect.top == lineRect.bottom )
			{
				lineRect.top-= 2;
				lineRect.bottom+= 2;
			}

			if( lineRect.top > lineRect.bottom )
			{
				int a = lineRect.bottom;
				lineRect.bottom = lineRect.top;
				lineRect.top = a;
			}

			if( rect.top > rect.bottom )
			{
				int a = rect.bottom;
				rect.bottom = rect.top;
				rect.top = a;
			}
```

```
                        if( lineRect.left > lineRect.right )
                        {
                                int a = lineRect.right;
                                lineRect.right = lineRect.left;
                                lineRect.left = a;
                        }

                        if( rect.left > rect.right )
                        {
                                int a = rect.right;
                                rect.right = rect.left;
                                rect.left = a;
                        }
                        if( lineRect.IntersectRect( &lineRect, &rect ))
                                return pObj;
                }
                else
                if (pObj->Intersects(rect))
                        return pObj;
        }
        return NULL;
}

void CPLCDoc::ComputePageSize( int x, int y )
{
        int nX, nY;
        if( !x ) nX = 850;
        else nX = x;

        if( !y ) nY = 4400;                     // 8.5" x 44" default
        else nY = y;

        CSize new_size( nX, nY );

        // if size changed then iterate over views and reset
        if (new_size != m_size)
        {
                m_size = new_size;
                POSITION pos = GetFirstViewPosition();
                while (pos != NULL)
                        ((CPLCView*)GetNextView(pos))->SetPageSize(m_size);
        }
}

void CPLCDoc::OnViewPaperColor()
{
        CColorDialog dlg;
        if (dlg.DoModal() != IDOK)
                return;

        m_paperColor = dlg.GetColor();
        SetModifiedFlag();
        UpdateAllViews(NULL);
}
```

```
// CPLCDoc diagnostics
#ifdef _DEBUG
void CPLCDoc::AssertValid() const
{
	COleDocument::AssertValid();
}

void CPLCDoc::Dump(CDumpContext& dc) const
{
	COleDocument::Dump(dc);
}
#endif //_DEBUG
```

```
// splitfrm.h : header file
```

```
// CSplitFrame frame with splitter
#ifndef __AFXEXT_H__
#include <afxext.h>
#endif
class CSplitFrame : public CMDIChildWnd
{
	DECLARE_DYNCREATE(CSplitFrame)
protected:
	CSplitFrame();		// protected constructor used by dynamic creation
	CSplitterWnd	m_wndSplitter;
public:
	virtual ~CSplitFrame();
	virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);

	// Generated message map functions
	//{{AFX_MSG(CSplitFrame)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
// splitfrm.cpp : implementation file
```

```
#include "stdafx.h"
#include "plcapp.h"
#include "splitfrm.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CSplitFrame

IMPLEMENT_DYNCREATE(CSplitFrame, CMDIChildWnd)

CSplitFrame::CSplitFrame()
{
```

```
}

CSplitFrame::~CSplitFrame()
{
}

BOOL CSplitFrame::OnCreateClient(LPCREATESTRUCT /*lpcs*/, CCreateContext* pContext)
{
	return m_wndSplitter.Create(this,
		2, 2,	// TODO: adjust the number of rows, columns
		CSize(10, 10),  // TODO: adjust the minimum pane size
		pContext);
}

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


/////////////////////////////////////////////////////////////////////////////
// cntritem.h : interface of the CDrawItem class
/////////////////////////////////////////////////////////////////////////////
class CPLCDoc;
class CPLCView;

class CDrawItem : public COleClientItem
{
	DECLARE_SERIAL(CDrawItem)
public:
	CDrawItem(CPLCDoc* pContainer = NULL, CDrawOleObj* pDrawObj = NULL);
		// Note: pContainer is allowed to be NULL to enable IMPLEMENT_SERIALIZE
		//  IMPLEMENT_SERIALIZE requires the class have a constructor with
		//  zero arguments.  Normally, OLE items are constructed with a
		//  non-NULL document pointer.

// Attributes
public:
	CPLCDoc* GetDocument()
		{ return (CPLCDoc*)COleClientItem::GetDocument(); }
	CPLCView* GetActiveView()
		{ return (CPLCView*)COleClientItem::GetActiveView(); }

	CDrawOleObj* m_pDrawObj;	// back pointer to OLE draw object
	BOOL UpdateExtent();
public:
	~CDrawItem();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	virtual void Serialize(CArchive& ar);
	virtual void OnGetItemPosition(CRect& rPosition);
```

```
protected:
	virtual void OnChange(OLE_NOTIFICATION wNotification, DWORD dwParam);
	virtual BOOL OnChangeItemPosition(const CRect& rectPos);
};

/////////////////////////////////////////////////////////////////////////////
// cntritem.cpp : interface of the CDrawItem class
/////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "plcapp.h"
#include "plcdoc.h"
#include "plcobj.h"
#include "plcview.h"
#include "cntritem.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CDrawItem implementation
IMPLEMENT_SERIAL(CDrawItem, COleClientItem, 0)
CDrawItem::CDrawItem(CPLCDoc* pContainer, CDrawOleObj* pDrawObj): OleClientItem(pContainer)
{
	m_pDrawObj = pDrawObj;
}

CDrawItem::~CDrawItem()
{
	if (m_pDrawObj != NULL)
		m_pDrawObj->m_pClientItem = NULL;
}

void CDrawItem::OnChange(OLE_NOTIFICATION nCode, DWORD dwParam)
{
	ASSERT_VALID(this);
	COleClientItem::OnChange(nCode, dwParam);
	switch(nCode)
	{
		case OLE_CHANGED_STATE:
		case OLE_CHANGED_ASPECT:
			m_pDrawObj->Invalidate();
			break;

		case OLE_CHANGED:
			UpdateExtent(); // extent may have changed
			m_pDrawObj->Invalidate();
			break;
	}
}

BOOL CDrawItem::OnChangeItemPosition(const CRect& rectPos)
{
	ASSERT_VALID(this);
```

```
	CPLCView* pView = GetActiveView();
	ASSERT_VALID(pView);
	CRect rect = rectPos;
	pView->ClientToDoc(rect);

	if (rect != m_pDrawObj->m_position)
	{
		m_pDrawObj->Invalidate();// invalidate old rectangle
		m_pDrawObj->m_position = rect; // update to new rectangle
		GetExtent(&m_pDrawObj->m_extent);
		m_pDrawObj->Invalidate(); // and invalidate new rectangle
		GetDocument()->SetModifiedFlag();// mark document as dirty
	}
	return COleClientItem::OnChangeItemPosition(rectPos);
}

void CDrawItem::OnGetItemPosition(CRect& rPosition)
{
	ASSERT_VALID(this);
	// update to extent of item if m_position is not initialized
	if (m_pDrawObj->m_position.IsRectEmpty())
		UpdateExtent();

	// copy m_position, which is in document coordinates
	CPLCView* pView = GetActiveView();
	ASSERT_VALID(pView);
	rPosition = m_pDrawObj->m_position;
	pView->DocToClient(rPosition);
}

void CDrawItem::Serialize(CArchive& ar)
{
	ASSERT_VALID(this);

	// Call base class first to read in COleClientItem data.
	// Note: this sets up the m_pDocument pointer returned from
	//  CDrawItem::GetDocument, therefore it is a good idea
	//  to call the base class Serialize first.
	COleClientItem::Serialize(ar);

	// now store/retrieve data specific to CDrawItem
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
}

BOOL CDrawItem::UpdateExtent()
{
	CSize size;
	if (!GetExtent(&size) || size == m_pDrawObj->m_extent)
		return FALSE;	// blank

	// if new object (i.e. m_extent is empty) setup position
```

```
	if (m_pDrawObj->m_extent == CSize(0, 0))
	{
		m_pDrawObj->m_position.right =
			m_pDrawObj->m_position.left + MulDiv(size.cx, 10, 254);
		m_pDrawObj->m_position.bottom =
			m_pDrawObj->m_position.top - MulDiv(size.cy, 10, 254);
	}
	// else data changed so scale up rect as well
	else if (!IsInPlaceActive() && size != m_pDrawObj->m_extent)
	{
		m_pDrawObj->m_position.right = m_pDrawObj->m_position.left +
			MulDiv(m_pDrawObj->m_position.Width(),
			size.cx, m_pDrawObj->m_extent.cx);

		m_pDrawObj->m_position.bottom = m_pDrawObj->m_position.top +
			MulDiv(m_pDrawObj->m_position.Height(),
			size.cy, m_pDrawObj->m_extent.cy);
	}

	m_pDrawObj->m_extent = size;
	m_pDrawObj->Invalidate();  // redraw to the new size/position
	return TRUE;
}

// CDrawItem diagnostics
#ifdef _DEBUG
void CDrawItem::AssertValid() const
{
	COleClientItem::AssertValid();
}

void CDrawItem::Dump(CDumpContext& dc) const
{
	COleClientItem::Dump(dc);
}
#endif
```

```
// RectDLG.h : defines classes to get user input for components
```

```
#ifndef _RECTDLG_H
#define _RECTDLG_H

#include "plcobj.h"
class CPLCView;
class CRectDlg : public CDialog
{
// Construction
public:
	CRectDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
	//{{AFX_DATA(CRectDlg)
	enum { IDD = IDD_PROP_RECT };
	BOOL	m_bNoFill;
```

```
	UINT	m_penSize;
	//}}AFX_DATA

// Implementation
protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support

	// Generated message map functions
	//{{AFX_MSG(CRectDlg)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

class CSetObjNameDlg : public CDialog
{
public:
	CSetObjNameDlg( CPLCComponentDrawTool* pTool, CDrawObj* pDrawObj,
		CPLCView* pParent = NULL);	// standard constructor
	LPCSTR GetValue1(){ return m_strValue; }
	LPCSTR GetValue2(){ return m_strValue2; }

// Dialog Data
	//{{AFX_DATA(CSetObjNameDlg)
	CEdit	m_editName;
	//}}AFX_DATA

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

	CString m_strValue;
	CString m_strValue2;

// Implementation
protected:
	// Generated message map functions
	//{{AFX_MSG(CSetObjNameDlg)
	virtual BOOL OnInitDialog();
	virtual void OnOK();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
	CPLCComponentDrawTool* m_pTool;
	CDrawObj* m_pDrawObj;
	BOOL CheckUniqueName( LPCSTR pszValue );
	BOOL CheckNonUniqueName();
private:
	BOOL ValidateInput( LPCSTR pszValue );
	BOOL ValidateNames( DrawShape  drawShape, LPCSTR pszValue, LPCSTR pszValue2 );
 };
#endif
```

```
// RectDLG.CPP : implementation file

#include "stdafx.h"
#include "plcapp.h"
#include "rectdlg.h"
#include "plcdoc.h"
#include "plcview.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CRectDlg dialog
CRectDlg::CRectDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CRectDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CRectDlg)
	m_bNoFill = FALSE;
	m_penSize = 0;
	//}}AFX_DATA_INIT
}

void CRectDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CRectDlg)
	DDX_Check(pDX, IDC_NOFILL, m_bNoFill);
	DDX_Text(pDX, IDC_WEIGHT, m_penSize);
	DDV_MinMaxUInt(pDX, m_penSize, 0, 32767);
	//}}AFX_DATA_MAP
}

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

// CRectDlg message handlers
// CSetObjNameDlg dialog
CSetObjNameDlg::CSetObjNameDlg( CPLCComponentDrawTool* pTool,
	CDrawObj* pDrawObj, CPLCView* pParent /*=NULL*/)
	: CDialog( pTool->m_nInputs == 1 ?
	IDD_DIALOG_OBJECT_NAME : IDD_DIALOG_OBJECT_NAMES, pParent)
{
	//{{AFX_DATA_INIT(CSetObjNameDlg)
	//}}AFX_DATA_INIT
	m_pTool = pTool;
	m_pDrawObj = pDrawObj;
	if( m_pDrawObj )
	{
		m_strValue = m_pDrawObj->m_strName;
		m_strValue2 = m_pDrawObj->m_strValue2;
	}
```

```
}

void CSetObjNameDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSetObjNameDlg)
	DDX_Control(pDX, IDC_EDIT_NAME, m_editName);
	//}}AFX_DATA_MAP
}

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

// CSetObjNameDlg message handlers

BOOL CSetObjNameDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	GetDlgItem( IDC_STATIC_VALUE )->SetWindowText( m_pTool->m_strInput1Title );
	SetWindowText( m_pTool->m_strDialogTitle );
	m_editName.SetWindowText( m_strValue );
	if( m_pTool->m_nInputs == 2 )
	{
		GetDlgItem( IDC_STATIC_VALUE2 )->SetWindowText
			( m_pTool->m_strInput2Title );
		GetDlgItem(IDC_EDIT_NAME2)->SetWindowText( m_strValue2 );
	}
	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

BOOL CSetObjNameDlg::ValidateInput( LPCSTR pszValue )
{
	int nLen = strlen( pszValue );
	int nCharsAllowed;
	if( m_pTool->m_nDecChars )
		nCharsAllowed = m_pTool->m_nDecChars;
	else
		nCharsAllowed = m_pTool->m_nHexChars;

	if( !nLen || nLen > nCharsAllowed )
	{
		CString strMsg, strMsg2;
		strMsg.Format( "Required %d %s char(s)",  m_pTool->m_nDecChars ?  m_pTool ->
			m_nDecChars : m_pTool->m_nHexChars,  m_pTool->m_nDecChars ?
			"Decimal" :"Hex" );

		AfxMessageBox( strMsg );
		return FALSE;
	}

	if( m_pTool->m_nHexChars > 0 )
```

```
	{
		for( int i = 0; i < nLen; i++ )
		{
			if( !isxdigit( pszValue[i] ) )
			{
				CString strMsg;
				strMsg.Format( "%c is not a Hex char", pszValue[i] );
				AfxMessageBox( strMsg );
				return FALSE;
			}
		}
	}
	if( m_pTool->m_nDecChars > 0 )
	{
		for( int i = 0; i < nLen; i++ )
		{
			if( !isdigit( pszValue[i] ) )
			{
				CString strMsg;
				strMsg.Format( "%c is not a Decimal number ", pszValue[i] );
				AfxMessageBox( strMsg );
				return FALSE;
			}
		}
	}
	return TRUE;
}

BOOL CSetObjNameDlg::CheckUniqueName( LPCSTR pszValue )
{
	if( !m_pTool )
		return TRUE;

	if(CheckNonUniqueName()) return TRUE;
	// check for unique number, parent is a CPLCView*
	CDrawObj* pObj = (((CPLCView*)m_pParentWnd )->GetDocument()->
		NameExists( pszValue, m_pTool->m_drawShape ));

	if( pObj && pObj != m_pDrawObj )
	{
		MessageBeep(MB_ICONHAND);
		AfxMessageBox( IDS_STRING_OBJECT_EXISTS );
		return FALSE;
	}
	else
		return TRUE;
}

BOOL CSetObjNameDlg::CheckNonUniqueName()
{
	if(m_pTool-> m_drawShape == greater_than || m_pTool-> m_drawShape == less_than ||
		 m_pTool-> m_drawShape == equal_to || m_pTool-> m_drawShape == plc_move)
		return TRUE;
	else
```

```
		return FALSE;
}

BOOL CSetObjNameDlg::ValidateNames( DrawShape  drawShape, LPCSTR pszValue, LPCSTR
	pszValue2 )
{
	switch( drawShape )
	{
		case serial_open :
		case parallel_open :
		case serial_closed :
		case parallel_closed :
		case plc_output :
		case reset_switch :
		case set_switch :
		case plc_counter :
		case plc_timer :
		case plc_start :
		case plc_end :
		case greater_than :
		case less_than :
		case equal_to :
		return TRUE;

		case plc_move :
		case plus :
		case minus :
			// both names should be different
			if( stricmp( pszValue, pszValue2 ) == 0 )
			{
				AfxMessageBox( "Source and Destination should be distinct" );
				return FALSE;
			}
			return TRUE;
	}
	ASSERT(FALSE);
	return FALSE;
}

void CSetObjNameDlg::OnOK()
{
	m_editName.GetWindowText( m_strValue );
	if( !ValidateInput( m_strValue ) || !CheckUniqueName(m_strValue) )
	{
		m_editName.SetFocus();
		return;
	}
	if( m_pTool->m_nInputs == 2 )
	{
		GetDlgItem(IDC_EDIT_NAME2)->GetWindowText( m_strValue2 );
		if( !ValidateInput( m_strValue2 ) )
		{
			GetDlgItem(IDC_EDIT_NAME2)->SetFocus();
			return;
```

```
		}
	}

	if( !ValidateNames( m_pTool->m_drawShape, m_strValue, m_strValue2 ) )
		return;

	CDialog::OnOK();
}
```

```
// PLCTool.h: interface for CPLCDrawTool and derivatives
```

```
#ifndef __PLCTOOL_H__
#define __PLCTOOL_H__

class CPLCView;
class CDrawRect;
extern enum DrawShape;
const int BITMAP_OBJ_DEFAULT_SIZE = 33;
const int BITMAP_OBJ_COUNTER_Y_SIZE = 35;
const int START_LINE_BEGIN_X = 20;
const int FIRST_COMPONENT_X_VALUE = 30;
const int CURSOR_X = 20;
const int START_LINE_BEGIN_Y = 50;
const int OVERLAP_WIDTH = 2;
const int DEFAULT_X_INCREMENT = 102;
const int START_LINE_END_Y  = 8000;
const int INTER_COMPONENT_SPACE = 30;
const int MINIMUM_DIST_BET_RUNG = 20;

class CDrawPoly;
class CDrawBitmap;
class CPLCDrawTool
{
public:
	CPLCDrawTool(DrawShape nDrawShape);
	virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);
	virtual void OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point);
	virtual void OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point);
	virtual void OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point);
	virtual void OnCancel();
	void TestConnection(CPLCView* pView);

// Attributes
	static CPLCDrawTool* FindTool(DrawShape drawShape);
	static CPtrList c_tools;
	static CPoint c_down;
	static UINT c_nDownFlags;
	static CPoint c_last;
	DrawShape m_drawShape;
	static DrawShape c_drawShape;
};

class CSelectTool : public CPLCDrawTool
{
```

```
public:
        CSelectTool();
        virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point);
        void CompleteRung(CPLCView* pView, UINT nFlags, const CPoint& point);
};

class CRectTool : public CPLCDrawTool
{
// Constructors
public:
        CRectTool(DrawShape drawShape);

// Implementation
        virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point);
};

class CPolyTool : public CPLCDrawTool
{
// Constructors
public:
        CPolyTool();

// Implementation
        virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnCancel();

        CDrawPoly* m_pDrawObj;
};

class CPLCComponentDrawTool : public CPLCDrawTool
{
        public:
        CPLCComponentDrawTool ( DrawShape nDrawShape ):
                CPLCDrawTool(nDrawShape) { m_pDrawObj = NULL; m_nInputs = 1; }
        virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point);
        virtual void OnCancel();
        //An instance of CDrawBitmap is defined and Parameter for the bitmap object are defined.
        //Parameter for the DialogBox are define.

        CDrawBitmap* m_pDrawObj;
        int m_nHexChars;  // if 0, no hex chars allowed
        int m_nDecChars;  // if 0, no decimal chars allowed
```

```
        int m_nBitmapID;
        int m_nBitmapWidth;
        int m_nBitmapHeight;
        int m_nInputs;
        int m_nConnectorHeight;
        int m_nConnectorWidth;
        CString m_strDialogTitle;
        CString m_strInput1Title;
        CString m_strInput2Title;
};

//Class for each Object is define which get the member
// and fuction for CPLCComponentDrawTool.
class CSerialOpenSwitch : public CPLCComponentDrawTool
{
// Constructors
        public:
                CSerialOpenSwitch();
};

class CSerialClosedSwitch : public CPLCComponentDrawTool
{
// Constructors
        public:
                CSerialClosedSwitch();
};

class CResetSwitch : public CPLCComponentDrawTool
{
// Constructors
        public:
                CResetSwitch();
};

class CPLCOutput : public CPLCComponentDrawTool
{
// Constructors
        public:
                CPLCOutput();
};

class CSetSwitch : public CPLCComponentDrawTool
{
// Constructors
        public:
                CSetSwitch();
};

class CSrcDestTool : public CPLCComponentDrawTool
{
        public:
        CSrcDestTool(DrawShape drawShape);
        virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);
```

```
        int m_nSrcBitmapOffset;
        int m_nDestBitmapOffset;
};

class CCompareTool : public CPLCComponentDrawTool
{
        public:
        CCompareTool(DrawShape drawShape);
        virtual void OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point);

        int m_nLineNumberOffset;
};

class CMove : public CSrcDestTool
{
// Constructors
        public:
                CMove();
};

class CPlus : public CSrcDestTool
{
// Constructors
        public:
                CPlus();
};

class CMinus : public CSrcDestTool
{
// Constructors
        public:
                CMinus();
};

class CPLCCounter: public CSrcDestTool
{
// Constructors
        public:
                CPLCCounter();
};

class CPLCTimer : public CSrcDestTool
{
// Constructors
        public:
                CPLCTimer();
};

class CGreaterThan : public CCompareTool
{
// Constructors
        public:
                CGreaterThan();
};
```

```
class CLessThan : public CCompareTool
{
// Constructors
        public:
                CLessThan();
};

class CEqualTo : public CCompareTool
{
// Constructors
        public:
                CEqualTo();
};

class CPLCStart: public CPLCComponentDrawTool
{
// Constructors
        public:
                CPLCStart();
};

class CPLCEnd : public CPLCComponentDrawTool
{
// Constructors
        public:
                CPLCEnd();
};

extern CSelectTool selectTool;
extern CRectTool lineTool;
extern CRectTool rectTool;
extern CRectTool roundRectTool;
extern CRectTool ellipseTool;
extern CPolyTool polyTool;

extern CSerialOpenSwitch  serialSwitch;
extern CSerialClosedSwitch serialClosedSwitch;
extern CResetSwitch resetSwitch;
extern CPLCOutput outputPLC;
extern CSetSwitch setSwitch;
extern CMove plcMove;
extern CGreaterThan plcGreater;
extern CLessThan plcLess;
extern CEqualTo plcEqual;
extern CPlus plcPlus;
extern CMinus plcMinus;
extern CPLCCounter      plcCounter;
extern CPLCTimer        plcTimer;
extern CPLCStartplcStart;
extern  CPLCEnd         plcEnd;
#endif // __PLCTOOL_H__
```

// PLCTool.CPP: implementation for drawing tools

```
#include "stdafx.h"
#include "plcapp.h"
#include "plcdoc.h"
#include "plcview.h"
#include "plcobj.h"
#include "plctool.h"
#include "mainfrm.h"
#include "math.h"

//CPtrList class supports ordered list of nonunique CObject
//pointer accessible sequentially.(doubly-Linked list).
CPtrList CPLCDrawTool::c_tools;

// global instances of tools
 CSelectTool selectTool;
 CRectTool lineTool(line);
 CRectTool rectTool(rect);
 CRectTool roundRectTool(roundRect);
 CRectTool ellipseTool(ellipse);
 CPolyTool polyTool;

 CSerialOpenSwitch  serialSwitch;
 CSerialClosedSwitch serialClosedSwitch;
 CResetSwitch resetSwitch;
 CPLCOutput outputPLC;
 CSetSwitch setSwitch;
 CMove plcMove;
 CGreaterThan plcGreater;
 CLessThan plcLess;
 CEqualTo plcEqual;
 CPlus plcPlus;
 CMinus plcMinus;
 CPLCCounter    plcCounter;
 CPLCTimer      plcTimer;
 CPLCStart      plcStart;
 CPLCEnd        plcEnd;

// CPLCDrawTool implementation
//Return the Cpoint which contain x,y coordinate of c_down
//c_down is the tool selected onlButtondown.
CPoint CPLCDrawTool::c_down;
UINT CPLCDrawTool::c_nDownFlags;

//Return the Cpoint which contain x,y coordinate of c_down
//c_down is the tool selected onlButtondown.
CPoint CPLCDrawTool::c_last;

//Return the DrawShape on selection (tool).
DrawShape CPLCDrawTool::c_drawShape = selection;

CPLCDrawTool::CPLCDrawTool(DrawShape drawShape)
{
```

```
	//In the contructor drawShape should be given,
	m_drawShape = drawShape;
	//and the tool which is pointed by c_tool is add at the tail.
	c_tools.AddTail(this);
}

//This method is use the CPLCVeiw.cpp to find tool
CPLCDrawTool* CPLCDrawTool::FindTool(DrawShape drawShape)
{
	POSITION pos = c_tools.GetHeadPosition();
	while (pos != NULL)
	{
		CPLCDrawTool* pTool = (CPLCDrawTool*)c_tools.GetNext(pos);
		if (pTool->m_drawShape == drawShape)
			return pTool;
	}
	return NULL;
}

// All the methods for mouse interaction with Screen(CView)
// for each tool are written below.
// For CPLCDrawTool::

void CPLCDrawTool::OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	// deactivate any in-place active item on this view!
	//Callthis fuction to get the OLE item that is currently
	// actived in place in the frame containing the view identified.
	COleClientItem* pActiveItem = pView->GetDocument()->
		GetInPlaceActiveItem(pView);
	if (pActiveItem != NULL)
	{
		pActiveItem->Close();
		ASSERT(pView->GetDocument()->
			GetInPlaceActiveItem(pView) == NULL);
	}

	//It is sent to the current CWnd object using the mouse input.
	pView->SetCapture();
	c_nDownFlags = nFlags;
	c_down = point;
	c_last = point;
}

void CPLCDrawTool::OnLButtonDblClk(CPLCView*, UINT, const CPoint&)
{
}

void CPLCDrawTool::OnLButtonUp(CPLCView*, UINT, const CPoint& point)
{
	// Releases the mouse capture from a current
	// CWnd object by using mouse input,
	ReleaseCapture();
```

```
	if (point == c_down)
		c_drawShape = selection;
}

void CPLCDrawTool::OnMouseMove(CPLCView*, UINT, const CPoint& point)
{
	c_last = point;
	SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
}

void CPLCDrawTool::OnCancel()
{
	c_drawShape = selection;
}

//For CSelect Tool::

CSelectTool::CSelectTool()
	: CPLCDrawTool(selection)
{
}

void CSelectTool::CompleteRung(CPLCView* pView,
	UINT nFlags, const CPoint& point)
{
	CPoint local = point;
	pView->ClientToDoc(local);
	CDrawObj* pObj = pView->GetDocument()->ObjectAt( local );
	if( !pObj || pObj->m_drawShape != line )
	{
		((CMainFrame*)AfxGetApp()->m_pMainWnd)->
			SetStatusMessage( IDS_STRING_BAD_POINT );
		MessageBeep(MB_ICONHAND);
		pView->m_selectMode = complete_rung;
		return;
	}

	CInstructRung* pUpperRung = ((CDrawRect*)pObj)->m_pRung;
	int pos;
	CInstruction* pInstruct = pUpperRung->
		InstructionAtLeft( pView, &pos, point.x );

	int nY = pView->GetCurrentLineY();
	int nLine = pView->GetCurrentLineNumber();
	CDrawRect* pLine = pView->GetCurrentLine();
	CInstructRung* pRung = pLine->m_pRung;

	if( !pRung || !pInstruct )
	{
		((CMainFrame*)AfxGetApp()->m_pMainWnd)->
			SetStatusMessage( IDS_STRING_BAD_POINT );
		MessageBeep(MB_ICONHAND);

		//CPLCDrawTool::c_drawShape = selection;
```

```
        pView->m_selectMode = complete_rung;
        return;
}

if( pRung->m_nCurrentX > point.x )
{
        ((CMainFrame*)AfxGetApp()->m_pMainWnd)->
                SetStatusMessage( IDS_STRING_BACKWARD_POINT );
        MessageBeep(MB_ICONHAND);
        //CPLCDrawTool::c_drawShape = selection;
        pView->m_selectMode = complete_rung;
        return;
}

pUpperRung->CompleteParallelRung( pRung->m_nRungNum, pos );

// extend horizontal line
{
        CRect rect = pLine->m_position;

        pView->DocToClient(rect);
        rect.right += point.x - pRung->m_nCurrentX;
        pView->ClientToDoc(rect);
        pLine->m_position = rect;
}

// put the cursor at the top of the line
CRect rect = pObj->m_position;
pView->DocToClient( rect );

// draw vertical line
pLine = pView->DrawLine( CRect( point.x, nY,
        point.x, rect.top ), CDrawRect::shunt, FALSE, FALSE  );
pRung->m_pParallelRungRightLine = pLine;

((CMainFrame*)AfxGetApp()->m_pMainWnd)->
        SetStatusMessage( AFX_IDS_IDLEMESSAGE );

if( pUpperRung )
{
        pView->SetCurrentXY( rect.top, point.x );
        pView->SetCursor( rect.top, point.x, pUpperRung );

        // draw all new objects after the completed rung
        // or after the last object on the rung

        if( point.x + INTER_COMPONENT_SPACE >
                pUpperRung->m_nNextObjectX )
                pUpperRung->m_nNextObjectX = point.x
                + INTER_COMPONENT_SPACE;

        pLine->m_pRung = pUpperRung;
}
else
```

```
		{
			pRung = pRung->m_pParentRung;
			pView->SetCurrentLine( (CDrawRect*)pObj );

			int nCurrentY = pObj->m_position.top;
			ASSERT( nCurrentY > 0 );

			pView->SetCurrentXY( nCurrentY , pRung->m_nNextObjectX );

			// put the cursor on the parent rung's current x position
			pView->SetCursor( nCurrentY, pRung->m_nNextObjectX, pRung );
			pLine->m_pRung = pRung;
		}
		pView->OnDrawSelect();
}

void CSelectTool::OnLButtonDown(CPLCView* pView,
		UINT nFlags, const CPoint& point)
{
		if( pView->m_selectMode == parallel_rung )
		{
			// must complete the rung first
			((CMainFrame*)AfxGetApp()->m_pMainWnd)->
					SetStatusMessage( IDS_INCOMPLETE_RUNG );
			MessageBeep(MB_ICONHAND);
			return;
		}

		if (pView->m_selectMode == complete_rung )
		{
			CompleteRung( pView, nFlags, point);
			return;
		}

		CPoint local = point;
		pView->ClientToDoc(local);
		CDrawObj* pObj;
		pView->m_selectMode = none;

		// Check for resizing (only allowed on single selections)
		if (pView->m_selection.GetCount() == 1)
		{
			pObj = pView->m_selection.GetHead();
			pView->m_nDragHandle = pObj->HitTest(local, pView, TRUE);
			if (pView->m_nDragHandle != 0)
				pView->m_selectMode = size;
		}

		// See if the click was on an object, select and start move if so
		if (pView->m_selectMode == none)
		{
			pObj = pView->GetDocument()->ObjectAt(local);

			if (pObj != NULL)
```

```
{
        pView->m_selectMode = move;

        // ctrl + click will select the object
        if (!pView->IsSelected(pObj) && (nFlags & MK_CONTROL) != 0 )
        {
                pView->Select(pObj, (nFlags & MK_SHIFT) != 0);
                return;
        }

        if( pObj->m_drawShape == line )
        {
                if( ((CDrawRect*)pObj)->m_LineType ==   CDrawRect::start ||
                        ((CDrawRect*)pObj)->m_LineType ==
                        CDrawRect::right_vertical || ((CDrawRect*)pObj)-
                        >m_LineType == CDrawRect::left_vertical ||
                        ((CDrawRect*)pObj)->m_LineType ==  CDrawRect::end )
                {
                        ((CMainFrame*)AfxGetApp()->m_pMainWnd)->
                                SetStatusMessage( IDS_BAD_CURSOR_POS );
                        MessageBeep(MB_ICONHAND);
                        return;
                }

                // put the cursor at the top of the line
                CRect rect = pObj->m_position;
                pView->DocToClient( rect );

                pView->SetCurrentXY( rect.top, point.x );
                CInstructRung* pRung =  ((CDrawRect*)pObj)->m_pRung;
                pView->SetCursor( rect.top, point.x, pRung );
        }
        else
        if( pObj->m_drawShape == serial_open ||
                pObj->m_drawShape == serial_closed )
        {
                // put the cursor on the line
                CRect rect = pObj->m_position;
                pView->DocToClient( rect );
                CPoint cursorPoint;
                if( rect.left - point.x > rect.right - point.x )
                {
                        // put the cursor to the right
                        cursorPoint.x = rect.right;
                }
                else
                {
                        // put the cursor to the left
                        cursorPoint.x = rect.left;
                }
                CPLCComponentDrawTool* pTool =
                        pObj->GetTool( pObj->m_drawShape );

                cursorPoint.y = rect.bottom - pTool->m_nConnectorHeight;
```

```
                        CInstructRung* pRung = pView->GetLadder()->GetRung( pObj );
                        if( pRung )
                        {
                                pView->SetCurrentXY( cursorPoint.y, cursorPoint.x );
                                pView->SetCursor( cursorPoint.y, cursorPoint.x, pRung );
                        }
                }

                // Ctrl+Click clones the selection...
                //if ((nFlags & MK_CONTROL) != 0)
                //      pView->CloneSelection();
            }
        }

        // Click on background, start a net-selection
        if (pView->m_selectMode == none)
        {
            if ((nFlags & MK_SHIFT) == 0)
                pView->Select(NULL);

            pView->m_selectMode = netSelect;

            CClientDC dc(pView);
            CRect rect(point.x, point.y, point.x, point.y);
            rect.NormalizeRect();
            dc.DrawFocusRect(rect);
        }

        pView->m_lastPoint = local;
        CPLCDrawTool::OnLButtonDown(pView, nFlags, point);
}

void CSelectTool::OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point)
{
        if ((nFlags & MK_SHIFT) != 0)
        {
            // Shift+DblClk deselects object...
            CPoint local = point;
            pView->ClientToDoc(local);
            CDrawObj* pObj = pView->GetDocument()->ObjectAt(local);
            if (pObj != NULL)
                pView->Deselect(pObj);
        }
        else
        {
            // "Normal" DblClk opens properties, or OLE server...
            if (pView->m_selection.GetCount() == 1)
                pView->m_selection.GetHead()->OnOpen(pView);
        }

        CPLCDrawTool::OnLButtonDblClk(pView, nFlags, point);
}
```

```
void CSelectTool::OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	if (pView->GetCapture() == pView)
	{
		if (pView->m_selectMode == netSelect)
		{
			CClientDC dc(pView);
			CRect rect(c_down.x, c_down.y, c_last.x, c_last.y);
			rect.NormalizeRect();
			dc.DrawFocusRect(rect);
			pView->SelectWithinRect(rect, TRUE);
		}
		else if (pView->m_selectMode != none)
		{
			// find out if the object is moved on a valid row
			if( pView->m_selection.GetCount() )
			{
				CDrawObj* pObj = pView->m_selection.GetHead();

				//pObj->m_position
			}

			pView->GetDocument()->UpdateAllViews(pView);
		}
	}
	CPLCDrawTool::OnLButtonUp(pView, nFlags, point);
}

void CSelectTool::OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	if (pView->GetCapture() != pView)
	{
		if (c_drawShape == selection && pView->m_selection.GetCount() == 1)
		{
			CDrawObj* pObj = pView->m_selection.GetHead();
			CPoint local = point;
			pView->ClientToDoc(local);
			int nHandle = pObj->HitTest(local, pView, TRUE);
			if (nHandle != 0)
			{
				SetCursor(pObj->GetHandleCursor(nHandle));
				return; // bypass CPLCDrawTool
			}
		}
		if (c_drawShape == selection)
			CPLCDrawTool::OnMouseMove(pView, nFlags, point);
		return;
	}

	if (pView->m_selectMode == netSelect)
	{
		CClientDC dc(pView);
		CRect rect(c_down.x, c_down.y, c_last.x, c_last.y);
		rect.NormalizeRect();
```

```
		dc.DrawFocusRect(rect);
		rect.SetRect(c_down.x, c_down.y, point.x, point.y);
		rect.NormalizeRect();
		dc.DrawFocusRect(rect);

		CPLCDrawTool::OnMouseMove(pView, nFlags, point);
		return;
	}

	CPoint local = point;
	pView->ClientToDoc(local);
	CPoint delta = (CPoint)(local - pView->m_lastPoint);

	POSITION pos = pView->m_selection.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = pView->m_selection.GetNext(pos);
		CRect position = pObj->m_position;

		if (pView->m_selectMode == move)
		{
			position += delta;

			//TRACE( "Move To: pObj->
			pObj->MoveTo(position, pView);
		}
		else if (pView->m_nDragHandle != 0)
		{
			pObj->MoveHandleTo(pView->m_nDragHandle, local, pView);
		}
	}

	pView->m_lastPoint = local;

	if (pView->m_selectMode == size && c_drawShape == selection)
	{
		c_last = point;
		SetCursor(pView->m_selection.GetHead()->
			GetHandleCursor(pView->m_nDragHandle));
		return; // bypass CPLCDrawTool
	}

	c_last = point;

	if (c_drawShape == selection)
		CPLCDrawTool::OnMouseMove(pView, nFlags, point);
}

//For CRectTool (does rectangles, round-rectangles, and ellipses)::

CRectTool::CRectTool(DrawShape drawShape)
	: CPLCDrawTool(drawShape)
{
}
```

```
void CRectTool::OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	CPLCDrawTool::OnLButtonDown(pView, nFlags, point);

	CPoint local = point;
	pView->ClientToDoc(local);

	CDrawRect* pObj = new CDrawRect(CRect(local, CSize(0, 0)), m_drawShape);
	switch (m_drawShape)
	{
		default:
			ASSERT(FALSE); // unsuported shape!
		case rect:
			pObj->m_nShape = CDrawRect::rectangle;
			break;
		case roundRect:
			pObj->m_nShape = CDrawRect::roundRectangle;
			break;
		case ellipse:
			pObj->m_nShape = CDrawRect::ellipse;
			break;
		case line:
			pObj->m_nShape = CDrawRect::line;
			pView->IncrNextLineNumber();
			pObj->m_nLineNumber = pView->GetNextLineNumber();
			break;
	}
	pView->GetDocument()->Add(pObj);
	pView->Select(pObj);

	pView->m_selectMode = size;
	pView->m_nDragHandle = 1;
	pView->m_lastPoint = local;
}

void CRectTool::OnLButtonDblClk(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	CPLCDrawTool::OnLButtonDblClk(pView, nFlags, point);
}

void CRectTool::OnLButtonUp(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	if (point == c_down)
	{
		CDrawObj *pObj = pView->m_selection.GetTail();// Don't create empty objects...
		pView->GetDocument()->Remove(pObj);
		pObj->Remove();
		selectTool.OnLButtonDown(pView, nFlags, point); // try a select!
	}
	selectTool.OnLButtonUp(pView, nFlags, point );
}

void CRectTool::OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point)
```

```
{
	SetCursor(AfxGetApp()->LoadStandardCursor(IDC_CROSS));
	selectTool.OnMouseMove(pView, nFlags, point);
}

// For CPolyTool::
CPolyTool::CPolyTool()
	: CPLCDrawTool(poly)
{
	m_pDrawObj = NULL;
}

void CPolyTool::OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	CPLCDrawTool::OnLButtonDown(pView, nFlags, point);
	CPoint local = point;
	pView->ClientToDoc(local);

	if (m_pDrawObj == NULL)
	{
		pView->SetCapture();

		m_pDrawObj = new CDrawPoly(CRect(local, CSize(0, 0)), m_drawShape);
		pView->GetDocument()->Add(m_pDrawObj);
		pView->Select(m_pDrawObj);
		m_pDrawObj->AddPoint(local, pView);
	}
	else if (local == m_pDrawObj->m_points[0])
	{
		// Stop when the first point is repeated...
		ReleaseCapture();
		m_pDrawObj->m_nPoints -= 1;
		if (m_pDrawObj->m_nPoints < 2)
		{
			m_pDrawObj->Remove();
		}
		else
		{
			pView->InvalObj(m_pDrawObj);
		}
		m_pDrawObj = NULL;
		c_drawShape = selection;
		return;
	}
	local.x += 1; // adjacent points can't be the same!
	m_pDrawObj->AddPoint(local, pView);

	pView->m_selectMode = size;
	pView->m_nDragHandle = m_pDrawObj->GetHandleCount();
	pView->m_lastPoint = local;
}

void CPolyTool::OnLButtonUp(CPLCView* , UINT , const CPoint& )
{
```

```
}

void CPolyTool::OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint& point)
{
	if (m_pDrawObj != NULL && (nFlags & MK_LBUTTON) != 0)
	{
		CPoint local = point;
		pView->ClientToDoc(local);
		m_pDrawObj->AddPoint(local);
		pView->m_nDragHandle = m_pDrawObj->GetHandleCount();
		pView->m_lastPoint = local;
		c_last = point;
		SetCursor(AfxGetApp()->LoadCursor(IDC_PENCIL));
	}
	else
	{
		SetCursor(AfxGetApp()->LoadStandardCursor(IDC_CROSS));
		selectTool.OnMouseMove(pView, nFlags, point);
	}
}

void CPolyTool::OnLButtonDblClk(CPLCView* pView, UINT , const CPoint& )
{
	ReleaseCapture();

	int nPoints = m_pDrawObj->m_nPoints;
	if (nPoints > 2 &&
		(m_pDrawObj->m_points[nPoints - 1] == m_pDrawObj->m_points[nPoints - 2] ||
		m_pDrawObj->m_points[nPoints - 1].x - 1 ==
			m_pDrawObj->m_points[nPoints - 2].x &&
		m_pDrawObj->m_points[nPoints - 1].y == m_pDrawObj->m_points[nPoints - 2].y))
	{
		// Nuke the last point if it's the same as the next to last...
		m_pDrawObj->m_nPoints -= 1;
		pView->InvalObj(m_pDrawObj);
	}
	m_pDrawObj = NULL;
	c_drawShape = selection;
}

void CPolyTool::OnCancel()
{
	CPLCDrawTool::OnCancel();
	m_pDrawObj = NULL;
}

CSerialOpenSwitch::CSerialOpenSwitch() : CPLCComponentDrawTool (serial_open)
{
	m_nHexChars = 2;
	m_nBitmapID = IDB_BITMAP_SERIAL_OPEN_SW;
	m_nBitmapWidth = 10;	//BITMAP_OBJ_DEFAULT_SIZE;
	m_nBitmapHeight = BITMAP_OBJ_DEFAULT_SIZE;

	m_nConnectorHeight = 22;//4;
```

```
	m_nConnectorWidth = 12;
	m_strDialogTitle = "Serial Open Contact";
	m_strInput1Title = "Enter Contact Number";
}

CSerialClosedSwitch::CSerialClosedSwitch()  : CPLCComponentDrawTool (serial_closed)
{
	m_nHexChars = 2;
	m_nBitmapID = IDB_BITMAP_SERIAL_CLOSED_SW;
	m_nBitmapWidth = BITMAP_OBJ_DEFAULT_SIZE;
	m_nBitmapHeight = BITMAP_OBJ_DEFAULT_SIZE;
	m_nConnectorHeight = 22;
	m_strDialogTitle = "Serial Closed Contact";
	m_strInput1Title = "Enter Contact Number";
}

CResetSwitch::CResetSwitch() : CPLCComponentDrawTool (reset_switch)
{
	m_nHexChars = 2;
	m_nBitmapID = IDB_BITMAP_RESET;
	m_nBitmapWidth = 48;
	m_nBitmapHeight = 35;
	m_nConnectorHeight = 25;
	m_nConnectorWidth = 4;

	m_strDialogTitle = "Reset Switch";
	m_strInput1Title = "Enter Switch Number";
}

CSetSwitch::CSetSwitch()  : CPLCComponentDrawTool (set_switch)
{
	m_nHexChars = 2;
	m_nBitmapID = IDB_BITMAP_SET;
	m_nBitmapWidth = 34;
	m_nBitmapHeight = 35;
	m_nConnectorHeight = 25;
	m_strDialogTitle = "Set Switch";
	m_strInput1Title = "Enter Switch Number";
}

CPLCOutput::CPLCOutput() : CPLCComponentDrawTool (plc_output)
{
	m_nHexChars = 2;
	m_nBitmapID = IDB_BITMAP_OUTPUT;
	m_nBitmapWidth = 20;
	m_nBitmapHeight = 34;
	m_nConnectorHeight = 23;
	m_nConnectorWidth = 7;
	m_strDialogTitle = "Coil Number";
	m_strInput1Title = "Enter Coil Number";
}

CSrcDestTool::CSrcDestTool(DrawShape drawShape) : CPLCComponentDrawTool(drawShape)
{
```

```
        m_nInputs = 2;
}

CMove::CMove() : CSrcDestTool(plc_move)
{
        m_nHexChars = 2;
        m_nBitmapID = IDB_BITMAP_MOVE;
        m_nBitmapWidth = 83;
        m_nBitmapHeight = 20;
        m_nSrcBitmapOffset = 36;
        m_nDestBitmapOffset = 60;
        m_nConnectorHeight = 9;
        m_nConnectorWidth = 4;
        m_strDialogTitle = "Move";
        m_strInput1Title = "Enter Source Contact";
        m_strInput2Title = "Enter Dest. Contact";
}

CPlus::CPlus() : CSrcDestTool(plus)
{
        m_nHexChars = 2;
        m_nBitmapID = IDB_BITMAP_ADDITION;
        m_nBitmapWidth = 65;
        m_nBitmapHeight = 20;
        m_nSrcBitmapOffset = 21;
        m_nDestBitmapOffset = 44;
        m_nConnectorHeight = 8;
        m_nConnectorWidth = 5;
        m_strDialogTitle = "Add";
        m_strInput1Title = "Enter Source Contact";
        m_strInput2Title = "Enter Dest. Contact";
}

CMinus::CMinus() : CSrcDestTool(minus)
{
        m_nHexChars = 2;
        m_nBitmapID = IDB_BITMAP_SUBTRACTION;
        m_nBitmapWidth = 65;
        m_nBitmapHeight = 20;
        m_nSrcBitmapOffset = 21;
        m_nDestBitmapOffset = 44;
        m_nConnectorHeight = 8;
        m_nConnectorWidth = 5;
        m_strDialogTitle = "Sub";
        m_strInput1Title = "Enter Source Contact";
        m_strInput2Title = "Enter Dest. Contact";
}

CCompareTool::CCompareTool(DrawShape drawShape) : CPLCComponentDrawTool(drawShape )
{
        m_nHexChars = 0;
        m_nDecChars = 3;

        m_nBitmapWidth = 52;
```

```
	m_nBitmapHeight = 20;
	m_nLineNumberOffset = 22;
	m_nConnectorHeight = 8;

	m_strInput1Title = "Enter Line Number";
}

CGreaterThan::CGreaterThan() : CCompareTool(greater_than)
{
	m_nBitmapID = IDB_BITMAP_GREATER_THAN;
	m_strDialogTitle = "Compare More Than";
	m_nConnectorWidth = 5;
}

CLessThan::CLessThan() : CCompareTool(less_than)
{
	m_nBitmapID = IDB_BITMAP_LESS_THAN;
	m_strDialogTitle = "Compare Less Than";
	m_nConnectorWidth = 5;
}

CEqualTo::CEqualTo() : CCompareTool(equal_to)
{
	m_nBitmapID = IDB_BITMAP_EQUAL_TO;
	m_strDialogTitle = "Compare If Equal";
	m_nConnectorWidth = 5;
}

CPLCCounter::CPLCCounter() : CSrcDestTool(plc_counter)
{
	m_nHexChars = 4;
	m_nBitmapID = IDB_BITMAP_COUNTER;
	m_nBitmapWidth = 41;
	m_nBitmapHeight = BITMAP_OBJ_COUNTER_Y_SIZE ;
	m_nSrcBitmapOffset = 4;
	m_nConnectorHeight = 36;
	m_nConnectorWidth = 3;
	m_strDialogTitle = "Counter";
	m_strInput1Title = "Enter Counter Number";
	m_strInput2Title = "Enter Number of Counts";
	m_nInputs = 2;
}

CPLCTimer::CPLCTimer() : CSrcDestTool(plc_timer)
{
	m_nHexChars = 4;
	m_nBitmapID = IDB_BITMAP_TIMER;
	m_nBitmapWidth =  41;
	m_nBitmapHeight = BITMAP_OBJ_COUNTER_Y_SIZE ;
	m_nConnectorHeight = 36;
	m_nSrcBitmapOffset = 4;
	m_nConnectorWidth = 3;
	m_strDialogTitle = "Timer";
	m_strInput1Title = "Enter Timer Number";
```

```
        m_strInput2Title = "Enter Time Delay";
        m_nInputs = 2;
}

CPLCStart::CPLCStart()   : CPLCComponentDrawTool (plc_start)
{
        m_nHexChars = 2;
        m_nBitmapID = IDB_BITMAP_START;
        m_nBitmapWidth = 35;
        m_nBitmapHeight = 35;
        m_nConnectorHeight = 24;
        m_nConnectorWidth = 3;
        m_strDialogTitle = "PLC I/O Cards";
        m_strInput1Title = "Enter Number of Cards";
}

CPLCEnd::CPLCEnd() : CPLCComponentDrawTool (plc_end)
{
        m_nHexChars = 2;
        m_nBitmapID = IDB_BITMAP_END;
        m_nBitmapWidth = 28;
        m_nBitmapHeight = 34 ;
        m_nConnectorHeight = 25;
        m_nConnectorWidth = 3;
        m_nInputs = 0;
}

void CSrcDestTool ::OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point)
{
        CPLCComponentDrawTool ::OnLButtonDown(pView, nFlags, point);
}

void CCompareTool ::OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint& point)
{
        CPLCComponentDrawTool ::OnLButtonDown(pView, nFlags, point);
}

void CPLCComponentDrawTool ::OnLButtonDown(CPLCView* pView, UINT nFlags, const CPoint&
        point)
{
        CPLCDrawTool::OnLButtonDown(pView, nFlags, point);
        CPoint local = point;
        pView->ClientToDoc(local);
}

double Distance( CPoint& p1, CPoint& p2 )
{
        double dVal =  sqrt( (double)((p1.x - p2.x)*(p1.x - p2.x)) + double((p1.y - p2.y)*(p1.y - p2.y)));
        return  dVal;
}

void CPLCComponentDrawTool ::OnLButtonUp(CPLCView* pView, UINT, const CPoint& point)
{
        ReleaseCapture();
```

```
	if (point == c_down)
		c_drawShape = selection;
}

void CPLCComponentDrawTool ::OnMouseMove(CPLCView* pView, UINT nFlags, const CPoint&
	point)
{
	if (m_pDrawObj != NULL && (nFlags & MK_LBUTTON) != 0)
	{
		CPoint local = point;
		pView->ClientToDoc(local);
		pView->m_lastPoint = local;
		c_last = point;
		SetCursor(AfxGetApp()->LoadCursor(IDC_PENCIL));
	}
}

void CPLCComponentDrawTool ::OnLButtonDblClk(CPLCView* pView, UINT , const CPoint& )
{
	ReleaseCapture();
}

void CPLCComponentDrawTool ::OnCancel()
{
	CPLCDrawTool::OnCancel();

	m_pDrawObj = NULL;
}

/////////////////////////////////////////////////////////////////////////////
// PLCObj.h: interface for CDrawObj and derivatives
/////////////////////////////////////////////////////////////////////////////
#ifndef __PLCOBJ_H__
#define __PLCOBJ_H__

class CPLCView;
class CPLCDoc;
class CPLCComponentDrawTool;

#include "plctool.h"

enum DrawShape
{
	selection,
	line,
	shunt,
	upward_shunt,
	rect,
	roundRect,
	ellipse,
	poly,
	ole_object,
	cursor,
```

```
        serial_open,
        parallel_open,
        serial_closed,
        parallel_closed,
        plc_output,
        reset_switch,
        set_switch,
        plc_move,
        greater_than,
        less_than,
        equal_to,
        plus,
        minus,
        plc_counter,
        plc_timer,
        plc_start,
        plc_end
};

class CInstructRung;

// CDrawObj - base class for all 'drawable objects'
class CDrawObj : public CObject
{
protected:
        DECLARE_SERIAL(CDrawObj);
        CDrawObj();

// Constructors
public:
        CDrawObj(const CRect& position, DrawShape drawShape,
                BOOL bMoveable = TRUE, BOOL bDeleteable = TRUE,
                LPCSTR pszName = NULL, LPCSTR pszValue2 = NULL );

// Attributes
        CRect m_position;
        CPLCDoc* m_pDocument;

        DrawShape m_drawShape;
        BOOL m_bMoveable;
        BOOL m_bDeletable;
        int m_nObjectNumber;
        virtual int GetHandleCount();
        virtual CPoint GetHandle(int nHandle);
        CRect GetHandleRect(int nHandleID, CPLCView* pView);
        virtual HCURSOR GetHandleCursor(int nHandle);
        virtual void SetLineColor(COLORREF color);
        virtual void SetFillColor(COLORREF color);

// Operations
        virtual void Draw(CDC* pDC);
        enum TrackerState { normal, selected, active };
        virtual void DrawTracker(CDC* pDC, TrackerState state);
        virtual void MoveTo(const CRect& positon, CPLCView* pView = NULL);
```

```
        virtual int HitTest(CPoint point, CPLCView* pView, BOOL bSelected);
        virtual BOOL Intersects(const CRect& rect);
        virtual void MoveHandleTo(int nHandle, CPoint point, CPLCView* pView = NULL);
        virtual void OnOpen(CPLCView* pView);
        virtual CDrawObj* Clone(CPLCDoc* pDoc = NULL);
        virtual void Remove();
        void Invalidate();

// Implementation
public:
        virtual ~CDrawObj();
        virtual void Serialize(CArchive& ar);
#ifdef _DEBUG
        void AssertValid();
#endif
        static CPLCComponentDrawTool* GetTool( DrawShape drawShape );
        CString m_strName;
        CString m_strValue2;
        char m_cPrefix;
        // implementation data
protected:
        BOOL m_bPen;
        LOGPEN m_logpen;
        BOOL m_bBrush;
        LOGBRUSH m_logbrush;
};

// special 'list' class for this application (requires afxtempl.h)
typedef CTypedPtrList<CObList, CDrawObj*> CDrawObjList;

// specialized draw objects

class CDrawRect : public CDrawObj
{
protected:
        DECLARE_SERIAL(CDrawRect);
        CDrawRect();

public:
        CDrawRect(const CRect& position, DrawShape drawShape);

// Implementation
public:
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC);
        virtual int GetHandleCount();
        virtual CPoint GetHandle(int nHandle);
        virtual HCURSOR GetHandleCursor(int nHandle);
        virtual void MoveHandleTo(int nHandle, CPoint point, CPLCView* pView = NULL);
        virtual BOOL Intersects(const CRect& rect);
        virtual CDrawObj* Clone(CPLCDoc* pDoc);
        int GetShape(){ return m_nShape; }
        enum Shape { rectangle, roundRectangle, ellipse, line };
        int m_nLineNumber;
```

```
        CInstructRung* m_pRung;
        enum LineType{ start, end, rung, shunt, right_vertical, left_vertical };
        LineType m_LineType;

protected:
        Shape m_nShape;
        CPoint m_roundness; // for roundRect corners
        friend class CRectTool;
};

class CDrawPoly;

class CDrawPoly : public CDrawObj
{
protected:
        DECLARE_SERIAL(CDrawPoly);
        CDrawPoly();

public:
        CDrawPoly(const CRect& position, DrawShape drawShape );

// Operations
        void AddPoint(const CPoint& point, CPLCView* pView = NULL);
        BOOL RecalcBounds(CPLCView* pView = NULL);
public:
        virtual ~CDrawPoly();
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC);
        virtual void MoveTo(const CRect& position, CPLCView* pView = NULL);
        virtual int GetHandleCount();
        virtual CPoint GetHandle(int nHandle);
        virtual HCURSOR GetHandleCursor(int nHandle);
        virtual void MoveHandleTo(int nHandle, CPoint point, CPLCView* pView = NULL);
        virtual BOOL Intersects(const CRect& rect);
        virtual CDrawObj* Clone(CPLCDoc* pDoc);

protected:
        int m_nPoints;
        int m_nAllocPoints;
        CPoint* m_points;
        CDrawPoly* m_pDrawObj;

        friend class CPolyTool;
};

class CDrawItem;    // COleClientItem derived class

class CDrawOleObj : public CDrawObj
{
protected:
        DECLARE_SERIAL(CDrawOleObj);
        CDrawOleObj();

public:
```

```
        CDrawOleObj(const CRect& position  );

// Implementation
public:
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC);
        virtual CDrawObj* Clone(CPLCDoc* pDoc);
        virtual void OnOpen(CPLCView* pView);
        virtual void MoveTo(const CRect& positon, CPLCView* pView = NULL);
        virtual void Remove();
        virtual ~CDrawOleObj();

        static BOOL c_bShowItems;

        CDrawItem* m_pClientItem;
        CSize m_extent; // current extent is tracked separate from scaled position
};

// specialized draw objects
class CDrawBitmap : public CDrawObj
{
protected:
        DECLARE_SERIAL(CDrawBitmap);
        CDrawBitmap();

public:
        CDrawBitmap(const CRect& position, DrawShape drawShape,
                BOOL bMoveable = TRUE, BOOL bDeleteable = TRUE,
                LPCSTR pszValue = NULL, LPCSTR pszValue2 = NULL );

// Implementation
public:
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC);
        virtual BOOL Intersects(const CRect& rect);
        virtual CDrawObj* Clone(CPLCDoc* pDoc);
        void MoveHandleTo(int nHandle, CPoint point, CPLCView* pView){} // no resizing allowed
        int GetBitmapID(){ return GetTool( m_drawShape )->m_nBitmapID; }
        CDrawRect* m_pLine;

protected:
        //doesn't use pen and brush
        friend class CPLCDrawTool;
};

class CDrawSrcDestObj: public CDrawBitmap
{
        public:
                CDrawSrcDestObj();
                int m_nSrc;
                int m_nDest;

                virtual void Serialize(CArchive& ar);
};
```

```
class CDrawCompareObj: public CDrawBitmap
{
        public:
                CDrawCompareObj();
                int m_nLineNumber;

                virtual void Serialize(CArchive& ar);
};
#endif // __PLCOBJ_H__
```

// plcobj.cpp - implementation for drawing objects

```
#include "stdafx.h"
#include "plcapp.h"
#include "plcdoc.h"
#include "plcview.h"
#include "plcobj.h"
#include "cntritem.h"
#include "rectdlg.h"
#define DEFAULT_LINE_WIDTH 2

int gObjects = 1;

IMPLEMENT_SERIAL(CDrawObj, CObject, 0)

CDrawObj::CDrawObj()
{
        m_nObjectNumber = 0;
        m_cPrefix = 0;
}

CDrawObj::~CDrawObj()
{
}

CDrawObj::CDrawObj(const CRect& position, DrawShape drawShape,  BOOL bMoveable, BOOL
        bDeleteable, LPCSTR pszName, LPCSTR pszValue2 )
{
        m_position = position;
        m_nObjectNumber = 0;
        m_drawShape = drawShape;
        m_pDocument = NULL;

        m_bPen = TRUE;
        m_logpen.lopnStyle = PS_INSIDEFRAME;
        m_logpen.lopnWidth.x = DEFAULT_LINE_WIDTH;
        m_logpen.lopnWidth.y = DEFAULT_LINE_WIDTH;
        m_logpen.lopnColor = RGB(0, 0, 0);

        m_bBrush = TRUE;
        m_logbrush.lbStyle = BS_SOLID;
```

```
	m_logbrush.lbColor = RGB(192, 192, 192);
	m_logbrush.lbHatch = HS_HORIZONTAL;

	if( pszName )
		m_strName = pszName;
	else
		m_strName.Format( "O%d", gObjects++ );

	if( pszValue2 )
		m_strValue2 = pszValue2;
	m_bMoveable = bMoveable;
	m_bDeletable = bDeleteable;
}

void CDrawObj::Serialize(CArchive& ar)
{
	CObject::Serialize(ar);
	if (ar.IsStoring())
	{
		ar << m_position;
		ar << (WORD)m_bPen;
		ar.Write(&m_logpen, sizeof(LOGPEN));
		ar << (WORD)m_bBrush;
		ar.Write(&m_logbrush, sizeof(LOGBRUSH));
		ar << (WORD)m_drawShape;
		ar << m_strName;
		ar << m_strValue2;
		ar << (WORD)m_bMoveable;
		ar << (WORD)m_bDeletable;
		ar << (WORD)m_nObjectNumber;
	}
	else
	{
		// get the document back pointer from the archive
		m_pDocument = (CPLCDoc*)ar.m_pDocument;
		ASSERT_VALID(m_pDocument);
		ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CPLCDoc)));

		WORD wTemp;
		ar >> m_position;
		ar >> wTemp; m_bPen = (BOOL)wTemp;
		ar.Read(&m_logpen,sizeof(LOGPEN));
		ar >> wTemp; m_bBrush = (BOOL)wTemp;
		ar.Read(&m_logbrush, sizeof(LOGBRUSH));
		ar >> wTemp; m_drawShape = (DrawShape)wTemp;
		ar >> m_strName;
		ar >> m_strValue2;
		ar >> wTemp; m_bMoveable = wTemp;
		ar >> wTemp; m_bDeletable = wTemp;
		ar >> wTemp; m_nObjectNumber = wTemp;
	}
}

void CDrawObj::Remove()
```

```
{
	delete this;
}

void CDrawObj::Draw(CDC*)
{
}

void CDrawObj::DrawTracker(CDC* pDC, TrackerState state)
{
	ASSERT_VALID(this);

	switch (state)
	{
	case normal:
		break;

	case selected:
	case active:
		{
			int nHandleCount = GetHandleCount();
			for (int nHandle = 1; nHandle <= nHandleCount; nHandle += 1)
			{
				CPoint handle = GetHandle(nHandle);
				pDC->PatBlt(handle.x - 3, handle.y - 3, 7, 7, DSTINVERT);
			}
		}
		break;
	}
}

// position is in logical
void CDrawObj::MoveTo(const CRect& position, CPLCView* pView)
{
	ASSERT_VALID(this);
	if (position == m_position)
		return;
	if (pView == NULL)
	{
		Invalidate();
		m_position = position;
		Invalidate();
	}
	else
	{
		pView->InvalObj(this);
		m_position = position;
		pView->InvalObj(this);
	}
	m_pDocument->SetModifiedFlag();
}

// Note: if bSelected, hit-codes start at one for the top-left  and increment clockwise, 0 means no hit.
// If !bSelected, 0 = no hit, 1 = hit (anywhere)
```

```
// point is in logical coordinates
int CDrawObj::HitTest(CPoint point, CPLCView* pView, BOOL bSelected)
{
	ASSERT_VALID(this);
	ASSERT(pView != NULL);

	if (bSelected)
	{
		int nHandleCount = GetHandleCount();
		for (int nHandle = 1; nHandle <= nHandleCount; nHandle += 1)
		{
			// GetHandleRect returns in logical coords
			CRect rc = GetHandleRect(nHandle,pView);
			if (point.x >= rc.left && point.x < rc.right &&
				point.y <= rc.top && point.y > rc.bottom)
				return nHandle;
		}
	}
	else
	{
		if (point.x >= m_position.left && point.x < m_position.right &&
			point.y <= m_position.top && point.y > m_position.bottom)
			return 1;
	}
	return 0;
}

// rect must be in logical coordinates
BOOL CDrawObj::Intersects(const CRect& rect)
{
	ASSERT_VALID(this);

	CRect fixed = m_position;
	fixed.NormalizeRect();
	CRect rectT = rect;
	rectT.NormalizeRect();
	return !(rectT & fixed).IsRectEmpty();
}

int CDrawObj::GetHandleCount()
{
	ASSERT_VALID(this);
	return 8;
}

// returns logical coords of center of handle
CPoint CDrawObj::GetHandle(int nHandle)
{
	ASSERT_VALID(this);
	int x, y, xCenter, yCenter;

	// this gets the center regardless of left/right
	// and top/bottom ordering
	xCenter = m_position.left + m_position.Width() / 2;
```

```
        yCenter = m_position.top + m_position.Height() / 2;

        switch (nHandle)
        {
        default:
                ASSERT(FALSE);

        case 1:
                x = m_position.left;
                y = m_position.top;
                break;

        case 2:
                x = xCenter;
                y = m_position.top;
                break;

        case 3:
                x = m_position.right;
                y = m_position.top;
                break;

        case 4:
                x = m_position.right;
                y = yCenter;
                break;

        case 5:
                x = m_position.right;
                y = m_position.bottom;
                break;

        case 6:
                x = xCenter;
                y = m_position.bottom;
                break;

        case 7:
                x = m_position.left;
                y = m_position.bottom;
                break;

        case 8:
                x = m_position.left;
                y = yCenter;
                break;
        }

        return CPoint(x, y);
}

// return rectange of handle in logical coords
CRect CDrawObj::GetHandleRect(int nHandleID, CPLCView* pView)
{
```

```
	ASSERT_VALID(this);
	ASSERT(pView != NULL);

	CRect rect;
	// get the center of the handle in logical coords
	CPoint point = GetHandle(nHandleID);
	// convert to client/device coords
	pView->DocToClient(point);
	// return CRect of handle in device coords
	rect.SetRect(point.x-3, point.y-3, point.x+3, point.y+3);
	pView->ClientToDoc(rect);
	return rect;
}

HCURSOR CDrawObj::GetHandleCursor(int nHandle)
{
	ASSERT_VALID(this);

	LPCTSTR id;
	switch (nHandle)
	{
	default:
		ASSERT(FALSE);

	case 1:
	case 5:
		id = IDC_SIZENWSE;
		break;

	case 2:
	case 6:
		id = IDC_SIZENS;
		break;

	case 3:
	case 7:
		id = IDC_SIZENESW;
		break;

	case 4:
	case 8:
		id = IDC_SIZEWE;
		break;
	}
	return AfxGetApp()->LoadStandardCursor(id);
}

// point must be in logical
void CDrawObj::MoveHandleTo(int nHandle, CPoint point, CPLCView* pView)
{
	ASSERT_VALID(this);

	CRect position = m_position;
	switch (nHandle)
```

```
		{
		default:
			ASSERT(FALSE);

		case 1:
			position.left = point.x;
			position.top = point.y;
			break;

		case 2:
			position.top = point.y;
			break;

		case 3:
			position.right = point.x;
			position.top = point.y;
			break;

		case 4:
			position.right = point.x;
			break;

		case 5:
			position.right = point.x;
			position.bottom = point.y;
			break;

		case 6:
			position.bottom = point.y;
			break;

		case 7:
			position.left = point.x;
			position.bottom = point.y;
			break;

		case 8:
			position.left = point.x;
			break;
		}

		MoveTo(position, pView);
}

void CDrawObj::Invalidate()
{
		ASSERT_VALID(this);
		m_pDocument->UpdateAllViews(NULL, HINT_UPDATE_DRAWOBJ, this);
}

CDrawObj* CDrawObj::Clone(CPLCDoc* pDoc)
{
		ASSERT_VALID(this);
```

```
	CDrawObj* pClone = new CDrawObj(m_position, m_drawShape );
	pClone->m_bPen = m_bPen;
	pClone->m_logpen = m_logpen;
	pClone->m_bBrush = m_bBrush;
	pClone->m_logbrush = m_logbrush;
	ASSERT_VALID(pClone);

	if (pDoc != NULL)
		pDoc->Add(pClone);
	return pClone;
}

void CDrawObj::OnOpen( CPLCView* pView )
{
	ASSERT_VALID(this);

	if( m_drawShape == line  )
	{
		CRectDlg dlg;
		dlg.m_bNoFill = !m_bBrush;
		dlg.m_penSize = m_bPen ? m_logpen.lopnWidth.x : 0;

		if (dlg.DoModal() != IDOK)
			return;

		m_bBrush = !dlg.m_bNoFill;
		m_bPen = dlg.m_penSize > 0;
		if (m_bPen)
		{
			m_logpen.lopnWidth.x = dlg.m_penSize;
			m_logpen.lopnWidth.y = dlg.m_penSize;
		}
	}
	else
	{
		// bitmap objects
		CPLCComponentDrawTool* pTool = GetTool( m_drawShape );
		if( !pTool )
			return;

		CSetObjNameDlg dlg( pTool, this, pView );
		if (dlg.DoModal() == IDOK)
		{
			// get the new stuff from the user
			m_strName = dlg.GetValue1();
			m_strValue2 = dlg.GetValue2();
		}
		else
			return;
	}
	Invalidate();
	m_pDocument->SetModifiedFlag();
}
```

```
void CDrawObj::SetLineColor(COLORREF color)
{
	ASSERT_VALID(this);

	m_logpen.lopnColor = color;
	Invalidate();
	m_pDocument->SetModifiedFlag();
}

void CDrawObj::SetFillColor(COLORREF color)
{
	ASSERT_VALID(this);

	m_logbrush.lbColor = color;
	Invalidate();
	m_pDocument->SetModifiedFlag();
}

#ifdef _DEBUG
void CDrawObj::AssertValid()
{
	ASSERT(m_position.left <= m_position.right);
	ASSERT(m_position.bottom <= m_position.top);
}
#endif

// CDrawRect
IMPLEMENT_SERIAL(CDrawRect, CDrawObj, 0)

CDrawRect::CDrawRect()
{
	m_nLineNumber = -1;
	m_pRung = NULL;
}

CDrawRect::CDrawRect(const CRect& position, DrawShape drawShape )
	: CDrawObj(position, drawShape )
{
	ASSERT_VALID(this);

	m_nShape = rectangle;
	m_roundness.x = 16;
	m_roundness.y = 16;
	m_pRung = NULL;
}

void CDrawRect::Serialize(CArchive& ar)
{
	ASSERT_VALID(this);

	CDrawObj::Serialize(ar);
	if (ar.IsStoring())
	{
		ar << (WORD)m_nShape;
```

```
		ar << m_roundness;
		ar << (WORD)m_LineType;
		ar << (WORD)m_nLineNumber;
		if( m_pRung )
			ar << (DWORD)m_pRung->m_nRungNum;
		else
			ar << (DWORD)0;
	}
	else
	{
		WORD wTemp;
		ar >> wTemp; m_nShape = (Shape)wTemp;
		ar >> m_roundness;
		ar >> wTemp; m_LineType = (LineType)wTemp;
		ar >> wTemp; m_nLineNumber = wTemp;
		DWORD dwTemp; ar >> dwTemp; m_pRung = (CInstructRung*)dwTemp;
	}
}

void CDrawRect::Draw(CDC* pDC)
{
	ASSERT_VALID(this);

	CBrush brush;
	if (!brush.CreateBrushIndirect(&m_logbrush))
		return;
	CPen pen;
	if (!pen.CreatePenIndirect(&m_logpen))
		return;

	CBrush* pOldBrush;
	CPen* pOldPen;

	if (m_bBrush)
		pOldBrush = pDC->SelectObject(&brush);
	else
		pOldBrush = (CBrush*)pDC->SelectStockObject(NULL_BRUSH);

	if (m_bPen)
		pOldPen = pDC->SelectObject(&pen);
	else
		pOldPen = (CPen*)pDC->SelectStockObject(NULL_PEN);

	CRect rect = m_position;
	switch (m_nShape)
	{
	case rectangle:
		pDC->Rectangle(rect);
		break;

	case roundRectangle:
		pDC->RoundRect(rect, m_roundness);
		break;
```

```
        case ellipse:
                pDC->Ellipse(rect);
                break;

        case line:

                if (rect.top > rect.bottom)
                {
                        rect.top -= m_logpen.lopnWidth.y / 2;
                        rect.bottom += (m_logpen.lopnWidth.y + 1) / 2;
                }
                else
                if (rect.top < rect.bottom) //hbm
                {
                        rect.top += (m_logpen.lopnWidth.y + 1) / 2;
                        rect.bottom -= m_logpen.lopnWidth.y / 2;
                }

                if (rect.left > rect.right)
                {
                        rect.left -= m_logpen.lopnWidth.x / 2;
                        rect.right += (m_logpen.lopnWidth.x + 1) / 2;
                }
                else
                if (rect.left < rect.right) //hbm
                {
                        rect.left += (m_logpen.lopnWidth.x + 1) / 2;
                        rect.right -= m_logpen.lopnWidth.x / 2;
                }


                /*
                if( abs( rect.top - rect.bottom ) >  abs( rect.left - rect.right ) )
                {
                        // draw vertical
                        rect.left = rect.right;
                }
                else
                {
                        // draw horizontal
                        rect.top = rect.bottom ;
                }

                m_position = rect;
                */
                pDC->MoveTo(rect.TopLeft());
                pDC->LineTo(rect.BottomRight());
                break;
        }

        pDC->SelectObject(pOldBrush);
        pDC->SelectObject(pOldPen);
}
```

```
int CDrawRect::GetHandleCount()
{
	ASSERT_VALID(this);

	return m_nShape == line ? 2 :
		CDrawObj::GetHandleCount() + (m_nShape == roundRectangle);
}

// returns center of handle in logical coordinates
CPoint CDrawRect::GetHandle(int nHandle)
{
	ASSERT_VALID(this);

	if (m_nShape == line && nHandle == 2)
		nHandle = 5;
	else if (m_nShape == roundRectangle && nHandle == 9)
	{
		CRect rect = m_position;
		rect.NormalizeRect();
		CPoint point = rect.BottomRight();
		point.x -= m_roundness.x / 2;
		point.y -= m_roundness.y / 2;
		return point;
	}

	return CDrawObj::GetHandle(nHandle);
}

HCURSOR CDrawRect::GetHandleCursor(int nHandle)
{
	ASSERT_VALID(this);

	if (m_nShape == line && nHandle == 2)
		nHandle = 5;
	else if (m_nShape == roundRectangle && nHandle == 9)
		return AfxGetApp()->LoadStandardCursor(IDC_SIZEALL);
	return CDrawObj::GetHandleCursor(nHandle);
}

// point is in logical coordinates
void CDrawRect::MoveHandleTo(int nHandle, CPoint point, CPLCView* pView)
{
	ASSERT_VALID(this);

	if (m_nShape == line && nHandle == 2)
		nHandle = 5;
	else if (m_nShape == roundRectangle && nHandle == 9)
	{
		CRect rect = m_position;
		rect.NormalizeRect();
		if (point.x > rect.right - 1)
			point.x = rect.right - 1;
		else if (point.x < rect.left + rect.Width() / 2)
			point.x = rect.left + rect.Width() / 2;
```

```
		if (point.y > rect.bottom - 1)
			point.y = rect.bottom - 1;
		else if (point.y < rect.top + rect.Height() / 2)
			point.y = rect.top + rect.Height() / 2;
		m_roundness.x = 2 * (rect.right - point.x);
		m_roundness.y = 2 * (rect.bottom - point.y);
		m_pDocument->SetModifiedFlag();
		if (pView == NULL)
			Invalidate();
		else
			pView->InvalObj(this);
		return;
	}
	CDrawObj::MoveHandleTo(nHandle, point, pView);
}

// rect must be in logical coordinates
BOOL CDrawRect::Intersects(const CRect& rect)
{
	ASSERT_VALID(this);

	CRect rectT = rect;
	rectT.NormalizeRect();

	CRect fixed = m_position;
	fixed.NormalizeRect();
	if ((rectT & fixed).IsRectEmpty())
		return FALSE;

	CRgn rgn;
	switch (m_nShape)
	{
	case rectangle:
		return TRUE;

	case roundRectangle:
		rgn.CreateRoundRectRgn(fixed.left, fixed.top, fixed.right, fixed.bottom,
			m_roundness.x, m_roundness.y);
		break;

	case ellipse:
		rgn.CreateEllipticRgnIndirect(fixed);
		break;

	case line:
		{
			int x = (m_logpen.lopnWidth.x + 5) / 2;
			int y = (m_logpen.lopnWidth.y + 5) / 2;
			POINT points[4];
			points[0].x = fixed.left;
			points[0].y = fixed.top;
			points[1].x = fixed.left;
			points[1].y = fixed.top;
			points[2].x = fixed.right;
```

```
                points[2].y = fixed.bottom;
                points[3].x = fixed.right;
                points[3].y = fixed.bottom;

                if (fixed.left < fixed.right)
                {
                        points[0].x -= x;
                        points[1].x += x;
                        points[2].x += x;
                        points[3].x -= x;
                }
                else
                {
                        points[0].x += x;
                        points[1].x -= x;
                        points[2].x -= x;
                        points[3].x += x;
                }

                if (fixed.top < fixed.bottom)
                {
                        points[0].y -= y;
                        points[1].y += y;
                        points[2].y += y;
                        points[3].y -= y;
                }
                else
                {
                        points[0].y += y;
                        points[1].y -= y;
                        points[2].y -= y;
                        points[3].y += y;
                }
                rgn.CreatePolygonRgn(points, 4, ALTERNATE);
            }
            break;
        }
        return rgn.RectInRegion(fixed);
}

CDrawObj* CDrawRect::Clone(CPLCDoc* pDoc)
{
        ASSERT_VALID(this);

        CDrawRect* pClone = new CDrawRect(m_position, m_drawShape);
        pClone->m_bPen = m_bPen;
        pClone->m_logpen = m_logpen;
        pClone->m_bBrush = m_bBrush;
        pClone->m_logbrush = m_logbrush;
        pClone->m_nShape = m_nShape;
        pClone->m_roundness = m_roundness;
        ASSERT_VALID(pClone);

        if (pDoc != NULL)
```

```
		pDoc->Add(pClone);

	ASSERT_VALID(pClone);
	return pClone;
}

// CDrawPoly
IMPLEMENT_SERIAL(CDrawPoly, CDrawObj, 0)
CDrawPoly::CDrawPoly()
{
	m_points = NULL;
	m_nPoints = 0;
	m_nAllocPoints = 0;
}

CDrawPoly::CDrawPoly(const CRect& position, DrawShape drawShape )
	: CDrawObj(position, drawShape )
{
	m_points = NULL;
	m_nPoints = 0;
	m_nAllocPoints = 0;
	m_bPen = TRUE;
	m_bBrush = FALSE;
}

CDrawPoly::~CDrawPoly()
{
	if (m_points != NULL)
		delete[] m_points;
}

void CDrawPoly::Serialize( CArchive& ar )
{
	int i;
	CDrawObj::Serialize( ar );
	if( ar.IsStoring() )
	{
		ar << (WORD) m_nPoints;
		ar << (WORD) m_nAllocPoints;
		for (i = 0;i< m_nPoints; i++)
			ar << m_points[i];
	}
	else
	{
		WORD wTemp;
		ar >> wTemp; m_nPoints = wTemp;
		ar >> wTemp; m_nAllocPoints = wTemp;
		m_points = new CPoint[m_nAllocPoints];
		for (i = 0;i < m_nPoints; i++)
			ar >> m_points[i];
	}
}

void CDrawPoly::Draw(CDC* pDC)
```

```
{
	ASSERT_VALID(this);

	CBrush brush;
	if (!brush.CreateBrushIndirect(&m_logbrush))
		return;
	CPen pen;
	if (!pen.CreatePenIndirect(&m_logpen))
		return;

	CBrush* pOldBrush;
	CPen* pOldPen;

	if (m_bBrush)
		pOldBrush = pDC->SelectObject(&brush);
	else
		pOldBrush = (CBrush*)pDC->SelectStockObject(NULL_BRUSH);

	if (m_bPen)
		pOldPen = pDC->SelectObject(&pen);
	else
		pOldPen = (CPen*)pDC->SelectStockObject(NULL_PEN);

	pDC->Polygon(m_points, m_nPoints);

	pDC->SelectObject(pOldBrush);
	pDC->SelectObject(pOldPen);
}

// position must be in logical coordinates
void CDrawPoly::MoveTo(const CRect& position, CPLCView* pView)
{
	ASSERT_VALID(this);
	if (position == m_position)
		return;

	if (pView == NULL)
		Invalidate();
	else
		pView->InvalObj(this);

	for (int i = 0; i < m_nPoints; i += 1)
	{
		m_points[i].x += position.left - m_position.left;
		m_points[i].y += position.top - m_position.top;
	}

	m_position = position;

	if (pView == NULL)
		Invalidate();
	else
		pView->InvalObj(this);
	m_pDocument->SetModifiedFlag();
```

```
}

int CDrawPoly::GetHandleCount()
{
	return m_nPoints;
}

CPoint CDrawPoly::GetHandle(int nHandle)
{
	ASSERT_VALID(this);
	ASSERT(nHandle >= 1 && nHandle <= m_nPoints);
	return m_points[nHandle - 1];
}

HCURSOR CDrawPoly::GetHandleCursor(int )
{
	return AfxGetApp()->LoadStandardCursor(IDC_ARROW);
}

// point is in logical coordinates
void CDrawPoly::MoveHandleTo(int nHandle, CPoint point, CPLCView* pView)
{
	ASSERT_VALID(this);
	ASSERT(nHandle >= 1 && nHandle <= m_nPcints);
	if (m_points[nHandle - 1] == point)
		return;

	m_points[nHandle - 1] = point;
	RecalcBounds(pView);

	if (pView == NULL)
		Invalidate();
	else
		pView->InvalObj(this);
	m_pDocument->SetModifiedFlag();
}

// rect must be in logical coordinates
BOOL CDrawPoly::Intersects(const CRect& rect)
{
	ASSERT_VALID(this);
	CRgn rgn;
	rgn.CreatePolygonRgn(m_points, m_nPoints, ALTERNATE);
	return rgn.RectInRegion(rect);
}

CDrawObj* CDrawPoly::Clone(CPLCDoc* pDoc)
{
	ASSERT_VALID(this);

	CDrawPoly* pClone = new CDrawPoly(m_position, m_drawShape);
	pClone->m_bPen = m_bPen;
	pClone->m_logpen = m_logpen;
	pClone->m_bBrush = m_bBrush;
```

```
	pClone->m_logbrush = m_logbrush;
	pClone->m_points = new CPoint[m_nAllocPoints];
	memcpy(pClone->m_points, m_points, sizeof(CPoint) * m_nPoints);
	pClone->m_nAllocPoints = m_nAllocPoints;
	pClone->m_nPoints = m_nPoints;
	ASSERT_VALID(pClone);

	if (pDoc != NULL)
		pDoc->Add(pClone);

	ASSERT_VALID(pClone);
	return pClone;
}

// point is in logical coordinates
void CDrawPoly::AddPoint(const CPoint& point, CPLCView* pView)
{
	ASSERT_VALID(this);
	if (m_nPoints == m_nAllocPoints)
	{
		CPoint* newPoints = new CPoint[m_nAllocPoints + 10];
		if (m_points != NULL)
		{
			memcpy(newPoints, m_points, sizeof(CPoint) * m_nAllocPoints);
			delete[] m_points;
		}
		m_points = newPoints;
		m_nAllocPoints += 10;
	}

	if (m_nPoints == 0 || m_points[m_nPoints - 1] != point)
	{
		m_points[m_nPoints++] = point;
		if (!RecalcBounds(pView))
		{
			if (pView == NULL)
				Invalidate();
			else
				pView->InvalObj(this);
		}
		m_pDocument->SetModifiedFlag();
	}
}

BOOL CDrawPoly::RecalcBounds(CPLCView* pView)
{
	ASSERT_VALID(this);

	if (m_nPoints == 0)
		return FALSE;

	CRect bounds(m_points[0], CSize(0, 0));
	for (int i = 1; i < m_nPoints; ++i)
	{
```

```
		if (m_points[i].x < bounds.left)
			bounds.left = m_points[i].x;
		if (m_points[i].x > bounds.right)
			bounds.right = m_points[i].x;
		if (m_points[i].y < bounds.top)
			bounds.top = m_points[i].y;
		if (m_points[i].y > bounds.bottom)
			bounds.bottom = m_points[i].y;
	}

	if (bounds == m_position)
		return FALSE;

	if (pView == NULL)
		Invalidate();
	else
		pView->InvalObj(this);

	m_position = bounds;

	if (pView == NULL)
		Invalidate();
	else
		pView->InvalObj(this);

	return TRUE;
}

IMPLEMENT_SERIAL(CDrawOleObj, CDrawObj, 0)

BOOL CDrawOleObj::c_bShowItems = TRUE;

CDrawOleObj::CDrawOleObj() : m_extent(0,0)
{
	m_pClientItem = NULL;
}

CDrawOleObj::CDrawOleObj(const CRect& position )
	: CDrawObj(position, ole_object ), m_extent(0, 0)
{
	m_pClientItem = NULL;
}

CDrawOleObj::~CDrawOleObj()
{
	if (m_pClientItem != NULL)
	{
		m_pClientItem->Release();
		m_pClientItem = NULL;
	}
}

void CDrawOleObj::Remove()
{
```

```
	if (m_pClientItem != NULL)
	{
		m_pClientItem->Delete();
		m_pClientItem = NULL;
	}
	CDrawObj::Remove();
}

void CDrawOleObj::Serialize( CArchive& ar )
{
	ASSERT_VALID(this);

	CDrawObj::Serialize(ar);

	if (ar.IsStoring())
	{
		ar << m_extent;
		ar << m_pClientItem;
	}
	else
	{
		ar >> m_extent;
		ar >> m_pClientItem;
		m_pClientItem->m_pDrawObj = this;
	}
}

CDrawObj* CDrawOleObj::Clone(CPLCDoc* pDoc)
{
	ASSERT_VALID(this);

	AfxGetApp()->BeginWaitCursor();

	CDrawOleObj* pClone = NULL;
	CDrawItem* pItem = NULL;
	TRY
	{
		// perform a "deep copy" --
		// need to copy CDrawOleObj and the CDrawItem
		// that it points to.
		CDrawOleObj* pClone = new CDrawOleObj(m_position);
		CDrawItem* pItem = new CDrawItem(m_pDocument, pClone);
		if (!pItem->CreateCloneFrom(m_pClientItem))
			AfxThrowMemoryException();

		pClone->m_pClientItem = pItem;
		pClone->m_bPen = m_bPen;
		pClone->m_logpen = m_logpen;
		pClone->m_bBrush = m_bBrush;
		pClone->m_logbrush = m_logbrush;
		ASSERT_VALID(pClone);

		if (pDoc != NULL)
			pDoc->Add(pClone);
```

```
		}
		CATCH_ALL(e)
		{
			pItem->Delete();
			pClone->m_pClientItem = NULL;
			pClone->Remove();
			AfxGetApp()->EndWaitCursor();

			THROW_LAST();
		}
		END_CATCH_ALL

		AfxGetApp()->EndWaitCursor();
		return pClone;
}

void CDrawOleObj::Draw(CDC* pDC)
{
		ASSERT_VALID(this);

		CDrawItem* pItem = m_pClientItem;
		if (pItem != NULL)
		{
			// draw the OLE item itself
			pItem->Draw(pDC, m_position);

			// don't draw tracker in print preview or on printer
			if (!pDC->IsPrinting())
			{
				// use a CRectTracker to draw the standard effects
				CRectTracker tracker;
				tracker.m_rect = m_position;
				pDC->LPtoDP(tracker.m_rect);

				if (c_bShowItems)
				{
					// put correct border depending on item type
					if (pItem->GetType() == OT_LINK)
						tracker.m_nStyle |= CRectTracker::dottedLine;
					else
						tracker.m_nStyle |= CRectTracker::solidLine;
				}

				// put hatching over the item if it is currently open
				if (pItem->GetItemState() == COleClientItem::openState ||
					pItem->GetItemState() == COleClientItem::activeUIState)
				{
					tracker.m_nStyle |= CRectTracker::hatchInside;
				}
				tracker.Draw(pDC);
			}
		}
}
```

```
void CDrawOleObj::OnOpen(CPLCView* pView)
{
	AfxGetApp()->BeginWaitCursor();
	m_pClientItem->DoVerb(
		GetKeyState(VK_CONTROL) < 0 ? OLEIVERB_OPEN : OLEIVERB_PRIMARY,
		pView);
	AfxGetApp()->EndWaitCursor();
}

// position is in logical
void CDrawOleObj::MoveTo(const CRect& position, CPLCView* pView)
{
	ASSERT_VALID(this);

	if (position == m_position)
		return;

	// call base class to update position
	CDrawObj::MoveTo(position, pView);

	// update position of in-place editing session on position change
	if (m_pClientItem->IsInPlaceActive())
		m_pClientItem->SetItemRects();
}

// CDrawBitmap
IMPLEMENT_SERIAL(CDrawBitmap, CDrawObj, 0)

CDrawBitmap::CDrawBitmap()
{
	m_pLine = NULL;
}

CDrawBitmap::CDrawBitmap(const CRect& position, DrawShape drawShape,
	BOOL bMoveable, BOOL bDeleteable, LPCSTR pszValue,
	LPCSTR pszValue2 ) : CDrawObj(position, drawShape,
	bMoveable, bDeleteable, pszValue, pszValue2 )
{
	ASSERT_VALID(this);
	m_pLine = NULL;
}

void CDrawBitmap::Serialize(CArchive& ar)
{
	ASSERT_VALID(this);

	if (ar.IsStoring())
	{
		if( m_pLine )
			ar << (DWORD)m_pLine->m_nLineNumber;
		else
			ar << (DWORD)NULL;
	}
	else
```

```
	{
		DWORD dwTemp;
		ar >> dwTemp;
		// use the line number to search the line
		m_pLine = (CDrawRect*)dwTemp;
	}

	CDrawObj::Serialize(ar);
}

void CDrawSrcDestObj::Serialize(CArchive& ar)
{
	ASSERT_VALID(this);

	// SAVE and Fixup m_pLine

	CDrawBitmap::Serialize(ar);
	if (ar.IsStoring())
	{
		ar << (WORD) m_nSrc;
		ar << (WORD) m_nDest;
	}
	else
	{
		WORD wTemp;
		ar >> wTemp;
		m_nSrc = (int)wTemp;
		ar >> wTemp;
		m_nDest = (int)wTemp;
	}
}

void CDrawCompareObj::Serialize(CArchive& ar)
{
	ASSERT_VALID(this);

	CDrawBitmap::Serialize(ar);
	if (ar.IsStoring())
	{
		//ar << (WORD) m_nLineNumber;
	}
	else
	{
		WORD wTemp;
		ar >> wTemp;
		//m_nLineNumber = (int)wTemp;
	}
}

CPLCComponentDrawTool* CDrawObj::GetTool( DrawShape drawShape )
{
	switch( drawShape )
	{
		case serial_open : return &serialSwitch;
```

```
		case serial_closed : return &serialClosedSwitch;
		case plc_output : return &outputPLC;
		case reset_switch : return &resetSwitch;
		case set_switch : return &setSwitch;
		case plc_move : return &plcMove;
		case greater_than : return &plcGreater;
		case less_than : return &plcLess;
		case equal_to : return &plcEqual;
		case plus : return &plcPlus;
		case minus : return &plcMinus;
		case plc_counter : return &plcCounter;
		case plc_timer : return &plcTimer;
		case plc_start : return &plcStart;
		case plc_end : return &plcEnd;
		default: ASSERT(FALSE);
	}
	return NULL;
}

void CDrawBitmap::Draw(CDC* pDC)
{
	ASSERT_VALID(this);

	BOOL bRet;
	BITMAP		stBmp;
	CBitmap		oBitmap, *poBitmap;
	CDC oMemoryDC;
	VERIFY( oMemoryDC.CreateCompatibleDC( pDC ) );
	bRet = oBitmap.LoadBitmap( GetBitmapID() );
	if( bRet )
	{
		oBitmap.GetObject( sizeof( BITMAP ), &stBmp );
		poBitmap = oMemoryDC.SelectObject( &oBitmap );
		bRet = pDC->BitBlt( m_position.left, m_position.top,
			stBmp.bmWidth, stBmp.bmHeight, &oMemoryDC, 0, 0, SRCCOPY );
	}
	CRect rect = m_position;
	CSize oSize = pDC->GetTextExtent( m_strName, m_strName.GetLength() );
	VERIFY(oMemoryDC.DeleteDC( ));
	CPLCComponentDrawTool* pTool = GetTool( m_drawShape );

	switch( m_drawShape )
	{
		case plus :
		case minus :
		case plc_move :
			// show the two data in the bitmap
			pDC->TextOut( rect.left +
				((CSrcDestTool*)pTool)->m_nSrcBitmapOffset,
				rect.top + ((CSrcDestTool*)pTool)->
				m_nBitmapHeight -2, m_strName, m_strName.GetLength() );

			pDC->TextOut( rect.left + ((CSrcDestTool*)pTool)->
				m_nDestBitmapOffset, rect.top +
```

```
                        ((CSrcDestTool*)pTool)->m_nBitmapHeight -2,
                        m_strValue2, m_strValue2.GetLength() );
                break;

            case plc_counter :
            case plc_timer :
                {
                        CString strVal = m_strName + " " + m_strValue2;

                        // show the counter/timer data in the bitmap
                        pDC->TextOut( rect.left + ((CSrcDestTool*)pTool)->
                                m_nSrcBitmapOffset, rect.top +
                                ((CSrcDestTool*)pTool)->m_nBitmapHeight -2, strVal,
                                strVal.GetLength() );
                }
                break;

            case greater_than :
            case less_than :
            case equal_to :
                {
                        CString strVal = m_strName;

                        // show the counter/timer data in the bitmap
                        pDC->TextOut( rect.left + ((CCompareTool*)pTool)->
                                m_nLineNumberOffset, rect.top +
                                ((CSrcDestTool*)pTool)->m_nBitmapHeight -2,
                                strVal, strVal.GetLength() );
                }
                break;

            default :
                pDC->TextOut( rect.left +
                        ( rect.right - rect.left - oSize.cx )/2,
                        rect.top + 12, m_strName, m_strName.GetLength() );
        }
}

// rect must be in logical coordinates
BOOL CDrawBitmap::Intersects(const CRect& rect)
{
        ASSERT_VALID(this);

        CRect rectT = rect;
        rectT.NormalizeRect();

        CRect fixed = m_position;
        fixed.NormalizeRect();
        if ((rectT & fixed).IsRectEmpty())
                return FALSE;
        return TRUE;
}

CDrawObj* CDrawBitmap::Clone(CPLCDoc* pDoc)
```

```
{
	ASSERT_VALID(this);

	CDrawBitmap* pClone = new CDrawBitmap(m_position, m_drawShape );
	pClone->m_bPen = m_bPen;
	pClone->m_logpen = m_logpen;
	pClone->m_bBrush = m_bBrush;
	pClone->m_logbrush = m_logbrush;

	ASSERT_VALID(pClone);

	if (pDoc != NULL)
		pDoc->Add(pClone);

	ASSERT_VALID(pClone);
	return pClone;
}
```

// plcview.h

```
#ifndef _PLCVIEW_H
#define _PLCVIEW_H

// Hints for UpdateAllViews/OnUpdate
#define HINT_UPDATE_WINDOW     0
#define HINT_UPDATE_DRAWOBJ    1
#define HINT_UPDATE_SELECTION  2
#define HINT_DELETE_SELECTION  3
#define HINT_UPDATE_OLE_ITEMS  4

#define DEFAULT_LINE_DIST 50

class CPLCDoc;
class CDrawObj;
class CInstructDoc;
#include "instruct.h"

enum SelectMode
{
	none,
	netSelect,
	move,
	size,
	complete_rung,
	parallel_rung
};

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

```
// Attributes
public:
        CPLCDoc* GetDocument()
                { return (CPLCDoc*)m_pDocument; }
        void SetPageSize(CSize size);
        CRect GetInitialPosition();

// Operations Method to Update the View
public:
        //Converts device units into logical units Rect for the device context
        void DocToClient(CRect& rect);
        //Converts device units into logical units point for the device context
        void DocToClient(CPoint& point);
        //Converts logical units into device units point for the Document
        void ClientToDoc(CPoint& point);
        //Converts logical units into device units Rect for the Document
        void ClientToDoc(CRect& rect);
        // Check if the Doc item is select and Update the View
        void Select(CDrawObj* pObj, BOOL bAdd = FALSE);
        // Check if the DrawObjList item is select and Update the View
        void SelectWithinRect(CRect rect, BOOL bAdd = FALSE);
        void Deselect(CDrawObj* pObj); //Deselect the item
        void CloneSelection();      // copies object and adds it to the document
        // this will update the item rectangles by calling OnGetPosRect & OnGetClipRect.
        void UpdateActiveItem();
        void InvalObj(CDrawObj* pObj);              // handles CDrawOleObj objects
        void Remove(CDrawObj* pObj);                //Find the Object and remove it from the array
        void PasteNative(COleDataObject& dataObject);       //for COleDataObject
        void PasteEmbedded(COleDataObject& dataObject); //for COleDataObject
        void ResetDocPtr(){ m_pInstructDoc= NULL; }

        CDrawRect* DrawLine( CRect& r, CDrawRect::LineType type,
                BOOL bMoveable = TRUE, BOOL bDeleteable = TRUE );


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

        // state variables
        SelectMode m_selectMode;
        int m_nDragHandle;
        CPoint m_lastPoint;

        //Method to Update the View Window overridden to draw this view
        virtual void OnDraw(CDC* pDC);
        // Called by the framework when a view is activated or deactivated.
        virtual void OnActivateView(BOOL bActivate, CView* pActiveView, CView* pDeactiveView);
        // Hints for UpdateAllViews/OnUpdate
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
```

```
        //Prepare the device context by set it pixel and setting the window
        virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo);
        // do the scroll and update the position of any in-place active item
        virtual BOOL OnScrollBy(CSize sizeScroll, BOOL bDoScroll);
         //Creates Standard window by using structure CREATESTRUCT
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        void DrawGrid(CDC* pDC);  //Draw the Grid of the view
        static CLIPFORMAT m_cfDraw; // custom clipboard format
        CDrawObjList m_selection;
        BOOL m_bGrid;
        COLORREF m_gridColor;
        BOOL m_bActive; // is the view active?

protected:
        virtual void OnInitialUpdate(); // called first time after construct

        // Printing support
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);

        // OLE Container support
public:
        virtual BOOL IsSelected(const CObject* pDocItem) const;

        //Methods to Get or Set or modify the members  of the Document
        int GetNextHorizLineYpos();
        int GetNextLineNumber();
        int GetMaxY();
        int IncrNextLineNumber();
        int GetCurrentLineNumber();
        void SetCurrentLineNumber( int n );
        void IncrCurrentLineNumber();
        void SetCurrentLineY(int nNewY);
        void ChangeCurrentLineYBy(int nSize );
        void SetCurrentLineX(int nNewX);
        CDrawRect* GetPCursor();
        void SetPCursor( CDrawRect* p );
        int GetCurrentLineY();
        CDrawRect* GetCurrentLine();
        void SetCurrentLine( CDrawRect* pLine );
        CLadder*  GetLadder();
        void SetCursor( int nNewY, int nNewX, CInstructRung* pRung );
        void SetCurrentXY( int nNewY, int nNewX );
        void ResetCursor();
        BOOL CheckCursor();

        //Methods for Drawing the CDrawObj Objects
public:
        afx_msg void OnDrawSelect();
        BOOL DrawComponent( DrawShape shape = selection );

        //Get or Set or modify the members  of the Document
```

```
private:
        BOOL OutputSwitchPresent();
        BOOL EndRungPresent();
        void SetLastRungY( int nY );
        int GetLastRungY();
        void IncrLastRungY( int nY );

        //Method required to Edit a Ladder
private:
        CInstructDoc* m_pInstructDoc;
        BOOL ProcessEndInstr();
        void InsertRung( int nY, BOOL bVerticalOnly = FALSE );
        CString GetCodeStr( CInstruction* pInstruction );
        void ShowGeneratedCode( CInstructRung* pRungList,
                BOOL bShow, int nParallelRungNumber );
        void RemoveRungFromScreen();
        void SelectRung( CInstructRung* pRung );
        POSITION GetRungPosition( int nY );
        void MoveObjectsDown( POSITION rPos );
        BOOL MakeSpace( int nY, CInstructRung* pRung );


// Generated message map functions
protected:
        //{{AFX_MSG(CPLCView)
        afx_msg void OnFilePrint();
        afx_msg void OnInsertObject();
        afx_msg void OnCancelEdit();
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
        afx_msg void OnDrawRoundRect();
        afx_msg void OnDrawRect();
        afx_msg void OnDrawLine();
        afx_msg void OnDrawEllipse();
        afx_msg void OnDrawSerialOpenSwitch();
        afx_msg void OnUpdateSerialOpen(CCmdUI* pCmdUI);
        afx_msg void OnUpdateDrawEllipse(CCmdUI* pCmdUI);
        afx_msg void OnUpdateDrawLine(CCmdUI* pCmdUI);
        afx_msg void OnUpdateDrawRect(CCmdUI* pCmdUI);
        afx_msg void OnUpdateDrawRoundRect(CCmdUI* pCmdUI);
        afx_msg void OnUpdateDrawSelect(CCmdUI* pCmdUI);
        afx_msg void OnUpdateSingleSelect(CCmdUI* pCmdUI);
        afx_msg void OnEditSelectAll();
        afx_msg void OnEditClear();
        afx_msg void OnUpdateAnySelect(CCmdUI* pCmdUI);
        afx_msg void OnDrawPolygon();
        afx_msg void OnUpdateDrawPolygon(CCmdUI* pCmdUI);
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnViewGrid();
        afx_msg void OnUpdateViewGrid(CCmdUI* pCmdUI);
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
        afx_msg void OnObjectFillColor();
```

```
afx_msg void OnObjectLineColor();
afx_msg void OnObjectMoveBack();
afx_msg void OnObjectMoveForward();
afx_msg void OnObjectMoveToBack();
afx_msg void OnObjectMoveToFront();
afx_msg void OnEditCopy();
afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
afx_msg void OnEditCut();
afx_msg void OnUpdateEditCut(CCmdUI* pCmdUI);
afx_msg void OnEditPaste();
afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
afx_msg void OnSetFocus(CWnd* pOldWnd);
afx_msg void OnViewShowObjects();
afx_msg void OnUpdateViewShowObjects(CCmdUI* pCmdUI);
afx_msg void OnEditProperties();
afx_msg void OnUpdateEditProperties(CCmdUI* pCmdUI);
afx_msg void OnDestroy();
afx_msg void OnUpdateEditSelectAll(CCmdUI* pCmdUI);
afx_msg void OnDrawSerialClosedSwitch();
afx_msg void OnUpdateDrawSerialClosedSwitch(CCmdUI* pCmdUI);
afx_msg void OnDrawOutput();
afx_msg void OnUpdateDrawOutput(CCmdUI* pCmdUI);
afx_msg void OnDrawResetSwitch();
afx_msg void OnUpdateDrawResetSwitch(CCmdUI* pCmdUI);
afx_msg void OnDrawSetSwitch();
afx_msg void OnUpdateDrawSetSwitch(CCmdUI* pCmdUI);
afx_msg void OnDrawMove();
afx_msg void OnUpdateDrawMove(CCmdUI* pCmdUI);
afx_msg void OnDrawSubtraction();
afx_msg void OnUpdateDrawSubtraction(CCmdUI* pCmdUI);
afx_msg void OnDrawAddition();
afx_msg void OnUpdateDrawAddition(CCmdUI* pCmdUI);
afx_msg void OnDrawEqualto();
afx_msg void OnUpdateDrawEqualto(CCmdUI* pCmdUI);
afx_msg void OnDrawGreaterthan();
afx_msg void OnUpdateDrawGreaterthan(CCmdUI* pCmdUI);
afx_msg void OnUpdateDrawLessthan(CCmdUI* pCmdUI);
afx_msg void OnDrawLessthan();
afx_msg void OnDrawStart();
afx_msg void OnUpdateDrawStart(CCmdUI* pCmdUI);
afx_msg void OnDrawStop();
afx_msg void OnUpdateDrawStop(CCmdUI* pCmdUI);
afx_msg void OnDrawTimer();
afx_msg void OnUpdateDrawTimer(CCmdUI* pCmdUI);
afx_msg void OnDrawCounter();
afx_msg void OnUpdateDrawCounter(CCmdUI* pCmdUI);
afx_msg void OnPlcGeneratecode();
afx_msg void OnDrawShunt();
afx_msg void OnUpdateDrawShunt(CCmdUI* pCmdUI);
afx_msg void OnDrawUpwardShunt();
afx_msg void OnUpdateUpwardShunt(CCmdUI* pCmdUI);
afx_msg void OnEditDeleterung();
afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
//}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};
#endif

/////////////////////////////////////////////////////////////////////////////
// plcview.cpp: implementation of the CPLCView class
/////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include <afxpriv.h>

#include "plcapp.h"
#include "rectdlg.h"
#include "plcdoc.h"
#include "plcobj.h"
#include "cntritem.h"
#include "plcview.h"
#include "mainfrm.h"
#include "instruct.h"
#include "plctool.h"
#include "instrvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// private clipboard format (list of Draw objects)
CLIPFORMAT CPLCView::m_cfDraw =
	(CLIPFORMAT)::RegisterClipboardFormat(_T("PLC Draw Tool"));

// CPLCView
IMPLEMENT_DYNCREATE(CPLCView, CScrollView)

BEGIN_MESSAGE_MAP(CPLCView, CScrollView)
	//{{AFX_MSG_MAP(CPLCView)
	ON_COMMAND(ID_OLE_INSERT_NEW, OnInsertObject)
	ON_COMMAND(ID_CANCEL_EDIT, OnCancelEdit)
	ON_WM_LBUTTONDOWN()
	ON_WM_LBUTTONUP()
	ON_WM_MOUSEMOVE()
	ON_WM_LBUTTONDBLCLK()
	ON_COMMAND(ID_DRAW_ROUNDRECT, OnDrawRoundRect)
	ON_COMMAND(ID_DRAW_RECT, OnDrawRect)
	ON_COMMAND(ID_DRAW_LINE, OnDrawLine)
	ON_COMMAND(ID_DRAW_ELLIPSE, OnDrawEllipse)
	ON_COMMAND(ID_DRAW_SERIAL_OPEN_SWITCH, OnDrawSerialOpenSwitch)
	ON_UPDATE_COMMAND_UI(ID_DRAW_SERIAL_OPEN_SWITCH, OnUpdateSerialOpen)
	ON_UPDATE_COMMAND_UI(ID_DRAW_ELLIPSE, OnUpdateDrawEllipse)
	ON_UPDATE_COMMAND_UI(ID_DRAW_LINE, OnUpdateDrawLine)
	ON_UPDATE_COMMAND_UI(ID_DRAW_RECT, OnUpdateDrawRect)
	ON_UPDATE_COMMAND_UI(ID_DRAW_ROUNDRECT, OnUpdateDrawRoundRect)
	ON_UPDATE_COMMAND_UI(ID_DRAW_SELECT, OnUpdateDrawSelect)
	ON_UPDATE_COMMAND_UI(ID_OBJECT_MOVEFORWARD, OnUpdateSingleSelect)
	ON_COMMAND(ID_EDIT_SELECT_ALL, OnEditSelectAll)
```

```
ON_COMMAND(ID_EDIT_CLEAR, OnEditClear)
ON_UPDATE_COMMAND_UI(ID_EDIT_CLEAR, OnUpdateAnySelect)
ON_COMMAND(ID_DRAW_POLYGON, OnDrawPolygon)
ON_UPDATE_COMMAND_UI(ID_DRAW_POLYGON, OnUpdateDrawPolygon)
ON_WM_SIZE()
ON_COMMAND(ID_VIEW_GRID, OnViewGrid)
ON_UPDATE_COMMAND_UI(ID_VIEW_GRID, OnUpdateViewGrid)
ON_WM_ERASEBKGND()
ON_COMMAND(ID_OBJECT_FILLCOLOR, OnObjectFillColor)
ON_COMMAND(ID_OBJECT_LINECOLOR, OnObjectLineColor)
ON_COMMAND(ID_OBJECT_MOVEBACK, OnObjectMoveBack)
ON_COMMAND(ID_OBJECT_MOVEFORWARD, OnObjectMoveForward)
ON_COMMAND(ID_OBJECT_MOVETOBACK, OnObjectMoveToBack)
ON_COMMAND(ID_OBJECT_MOVETOFRONT, OnObjectMoveToFront)
ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
ON_COMMAND(ID_EDIT_CUT, OnEditCut)
ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateEditCut)
ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
ON_WM_SETFOCUS()
ON_COMMAND(ID_VIEW_SHOWOBJECTS, OnViewShowObjects)
ON_UPDATE_COMMAND_UI(ID_VIEW_SHOWOBJECTS, OnUpdateViewShowObjects)
ON_COMMAND(ID_EDIT_PROPERTIES, OnEditProperties)
ON_UPDATE_COMMAND_UI(ID_EDIT_PROPERTIES, OnUpdateEditProperties)
ON_WM_DESTROY()
ON_UPDATE_COMMAND_UI(ID_EDIT_SELECT_ALL, OnUpdateEditSelectAll)
ON_COMMAND(ID_DRAW_SERIAL_CLOSED_SWITCH, OnDrawSerialClosedSwitch)
ON_UPDATE_COMMAND_UI(ID_DRAW_SERIAL_CLOSED_SWITCH,
        OnUpdateDrawSerialClosedSwitch)
ON_COMMAND(ID_DRAW_OUTPUT, OnDrawOutput)
ON_UPDATE_COMMAND_UI(ID_DRAW_OUTPUT, OnUpdateDrawOutput)
ON_COMMAND(ID_DRAW_RESET_SWITCH, OnDrawResetSwitch)
ON_UPDATE_COMMAND_UI(ID_DRAW_RESET_SWITCH, OnUpdateDrawResetSwitch)
ON_COMMAND(ID_DRAW_SET_SWITCH, OnDrawSetSwitch)
ON_UPDATE_COMMAND_UI(ID_DRAW_SET_SWITCH, OnUpdateDrawSetSwitch)
ON_COMMAND(ID_DRAW_MOVE, OnDrawMove)
ON_UPDATE_COMMAND_UI(ID_DRAW_MOVE, OnUpdateDrawMove)
ON_COMMAND(ID_DRAW_SUBTRACTION, OnDrawSubtraction)
ON_UPDATE_COMMAND_UI(ID_DRAW_SUBTRACTION, OnUpdateDrawSubtraction)
ON_COMMAND(ID_DRAW_ADDITION, OnDrawAddition)
ON_UPDATE_COMMAND_UI(ID_DRAW_ADDITION, OnUpdateDrawAddition)
ON_COMMAND(ID_DRAW_EQUALTO, OnDrawEqualto)
ON_UPDATE_COMMAND_UI(ID_DRAW_EQUALTO, OnUpdateDrawEqualto)
ON_COMMAND(ID_DRAW_GREATERTHAN, OnDrawGreaterthan)
ON_UPDATE_COMMAND_UI(ID_DRAW_GREATERTHAN, OnUpdateDrawGreaterthan)
ON_UPDATE_COMMAND_UI(ID_DRAW_LESSTHAN, OnUpdateDrawLessthan)
ON_COMMAND(ID_DRAW_LESSTHAN, OnDrawLessthan)
ON_COMMAND(ID_DRAW_START, OnDrawStart)
ON_UPDATE_COMMAND_UI(ID_DRAW_START, OnUpdateDrawStart)
ON_COMMAND(ID_DRAW_STOP, OnDrawStop)
ON_UPDATE_COMMAND_UI(ID_DRAW_STOP, OnUpdateDrawStop)
ON_COMMAND(ID_DRAW_TIMER, OnDrawTimer)
ON_UPDATE_COMMAND_UI(ID_DRAW_TIMER, OnUpdateDrawTimer)
```

```
	ON_COMMAND(ID_DRAW_COUNTER, OnDrawCounter)
	ON_UPDATE_COMMAND_UI(ID_DRAW_COUNTER, OnUpdateDrawCounter)
	ON_COMMAND(ID_PLC_GENERATECODE, OnPlcGeneratecode)
	ON_COMMAND(ID_DRAW_SHUNT, OnDrawShunt)
	ON_UPDATE_COMMAND_UI(ID_DRAW_SHUNT, OnUpdateDrawShunt)
	ON_COMMAND(ID_DRAW_UPWARD_SHUNT, OnDrawUpwardShunt)
	ON_UPDATE_COMMAND_UI(ID_DRAW_UPWARD_SHUNT, OnUpdateUpwardShunt)
	ON_COMMAND(ID_EDIT_DELETERUNG, OnEditDeleterung)
	ON_UPDATE_COMMAND_UI(ID_OBJECT_MOVETOBACK, OnUpdateSingleSelect)
	ON_UPDATE_COMMAND_UI(ID_OBJECT_MOVETOFRONT, OnUpdateSingleSelect)
	ON_UPDATE_COMMAND_UI(ID_OBJECT_MOVEBACK, OnUpdateSingleSelect)
	ON_COMMAND(ID_DRAW_SELECT, OnDrawSelect)
	ON_WM_VSCROLL()
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

// CPLCView construction/destruction

CPLCView::CPLCView()
{
	m_bGrid = TRUE;
	m_gridColor = RGB(0, 0, 128);
	m_bActive = FALSE;
	m_pInstructDoc = NULL;
	m_selectMode = none;
}

CPLCView::~CPLCView()
{
}

BOOL CPLCView::PreCreateWindow(CREATESTRUCT& cs)
{
	ASSERT(cs.style & WS_CHILD);
	if (cs.lpszClass == NULL)
		cs.lpszClass = AfxRegisterWndClass(CS_DBLCLKS);
	return TRUE;
}

void CPLCView::OnActivateView(BOOL bActivate, CView* pActiveView,
	CView* pDeactiveView)
{
	CView::OnActivateView(bActivate, pActiveView, pDeactiveView);

	// invalidate selections when active status changes
	if (m_bActive != bActivate)
	{
		if (bActivate)  // if becoming active update as if active
			m_bActive = bActivate;
		if (!m_selection.IsEmpty())
			OnUpdate(NULL, HINT_UPDATE_SELECTION, NULL);
```

```
		m_bActive = bActivate;
	}
}

void CPLCView::SetCurrentXY( int nNewY, int nNewX )
{
	SetCurrentLineY(nNewY);
	SetCurrentLineX(nNewX);

	CPoint linePoint( nNewX, nNewY );
	ClientToDoc(linePoint);
	CDrawObj* pObj = GetDocument()->ObjectAt(linePoint);
	if( !pObj ) // this might be a shunt
	{
		// don't do anything
	}
	else
	if( pObj->m_drawShape == line )
	{
		SetCurrentLine( (CDrawRect*)pObj );
		ASSERT( ((CDrawRect*)pObj)->m_nLineNumber != -1 );
	}
	else
	{
		if( pObj->m_drawShape == rect )
			SetCurrentLine( GetDocument()->m_cursor.m_pCurrentLine );
		else
		{
			ASSERT( pObj->m_drawShape == serial_open ||
				pObj->m_drawShape == serial_closed );
			SetCurrentLine( ((CDrawBitmap*)pObj)->m_pLine );
		}
	}
}

void CPLCView::SetCursor( int nNewY, int nNewX, CInstructRung* pRung )
{
	if( !GetPCursor() )
	{
		CPoint local( CURSOR_X, START_LINE_BEGIN_Y );
		ClientToDoc(local);
		CDrawRect* pCursor = new CDrawRect(CRect(local, CSize(8, 8)), rect );
		SetPCursor( pCursor );
		GetDocument()->Add( pCursor );
		pCursor->m_bMoveable = FALSE;
		pCursor->m_bDeletable = FALSE;
	}

	if( pRung ) // set the current X
		pRung->m_nCurrentX = nNewX;

	CPoint local( nNewX, nNewY );
	ClientToDoc(local);
	GetPCursor()->m_position = CRect(local, CSize(8, 8));
```

```
		// bring cursor to front, unselect everything
		m_selection.RemoveAll();
		m_selection.AddTail(GetPCursor());

		// bring cursor to front
		OnObjectMoveToFront();
		m_selection.RemoveAll();
		Invalidate();
}

// CPLCView drawing
void CPLCView::InvalObj(CDrawObj* pObj)
{
		CRect rect = pObj->m_position;
		DocToClient(rect);
		if (m_bActive && IsSelected(pObj))
		{
			rect.left -= 4;
			rect.top -= 5;
			rect.right += 5;
			rect.bottom += 4;
		}
		rect.InflateRect(1, 1); // handles CDrawOleObj objects
		InvalidateRect(rect, FALSE);
}

void CPLCView::OnUpdate(CView* , LPARAM lHint, CObject* pHint)
{
		switch (lHint)
		{
		case HINT_UPDATE_WINDOW:	// redraw entire window
			Invalidate(FALSE);
			break;
		case HINT_UPDATE_DRAWOBJ:	// a single object has changed
			InvalObj((CDrawObj*)pHint);
			break;
		case HINT_UPDATE_SELECTION: // an entire selection has changed
			{
				CDrawObjList* pList = pHint != NULL ?
					(CDrawObjList*)pHint : &m_selection;
				POSITION pos = pList->GetHeadPosition();
				while (pos != NULL)
					InvalObj(pList->GetNext(pos));
			}
			break;
		case HINT_DELETE_SELECTION: // an entire selection has been removed
			if (pHint != &m_selection)
			{
				CDrawObjList* pList = (CDrawObjList*)pHint;
				POSITION pos = pList->GetHeadPosition();
				while (pos != NULL)
				{
					CDrawObj* pObj = pList->GetNext(pos);
```

```
                        if( pObj->m_drawShape == plc_start )
                                continue;

                        InvalObj(pObj);
                        Remove(pObj);   // remove it from this view's selection
                }
        }
        break;
    case HINT_UPDATE_OLE_ITEMS:
        {
                CPLCDoc* pDoc = GetDocument();
                POSITION pos = pDoc->GetObjects()->GetHeadPosition();
                while (pos != NULL)
                {
                        CDrawObj* pObj = pDoc->GetObjects()->GetNext(pos);
                        if (pObj->IsKindOf(RUNTIME_CLASS(CDrawOleObj)))
                                InvalObj(pObj);
                }
        }
        break;

    default:
        ASSERT(FALSE);
        break;
    }
}

void CPLCView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    CScrollView::OnPrepareDC(pDC, pInfo);
    // mapping mode is MM_ANISOTROPIC
    // these extents setup a mode similar to  MM_LOENGLISH
    // MM_LOENGLISH is in .01 physical inches
    // these extents provide .01 logical inches

    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetViewportExt(pDC->GetDeviceCaps(LOGPIXELSX),
        pDC->GetDeviceCaps(LOGPIXELSY));
    pDC->SetWindowExt(100, -100);

    // set the origin of the coordinate system to the center of the page
    CPoint ptOrg;
    ptOrg.x = GetDocument()->GetSize().cx / 2;
    ptOrg.y = GetDocument()->GetSize().cy / 2;
    // ptOrg is in logical coordinates
    pDC->OffsetWindowOrg(-ptOrg.x,ptOrg.y);
}

BOOL CPLCView::OnScrollBy(CSize sizeScroll, BOOL bDoScroll)
{
    // do the scroll
    if (!CScrollView::OnScrollBy(sizeScroll, bDoScroll))
        return FALSE;
```

```
	// update the position of any in-place active item
	if (bDoScroll)
	{
		UpdateActiveItem();
		UpdateWindow();
	}
	return TRUE;
}

void CPLCView::OnDraw(CDC* pDC)
{
	CPLCDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	CDC dc;
	CDC* pDrawDC = pDC;
	CBitmap bitmap;
	CBitmap* pOldBitmap;

	// only paint the rect that needs repainting
	CRect client;
	pDC->GetClipBox(client);
	CRect rect = client;
	DocToClient(rect);
	if (!pDC->IsPrinting())
	{
		// draw to offscreen bitmap for fast looking repaints
		if (dc.CreateCompatibleDC(pDC))
		{
			if (bitmap.CreateCompatibleBitmap(pDC, rect.Width(), rect.Height()))
			{
				OnPrepareDC(&dc, NULL);
				pDrawDC = &dc;

				// offset origin more because bitmap is just piece of the whole drawing
				dc.OffsetViewportOrg(-rect.left, -rect.top);
				pOldBitmap = dc.SelectObject(&bitmap);
				dc.SetBrushOrg(rect.left % 8, rect.top % 8);

				// might as well clip to the same rectangle
				dc.IntersectClipRect(client);
			}
		}
	}

	// paint background
	CBrush brush;
	if (!brush.CreateSolidBrush(pDoc->GetPaperColor()))
		return;

	brush.UnrealizeObject();
	pDrawDC->FillRect(client, &brush);
```

```
	if (!pDC->IsPrinting() && m_bGrid)
		DrawGrid(pDrawDC);

	pDoc->Draw(pDrawDC, this);

	if (pDrawDC != pDC)
	{
		pDC->SetViewportOrg(0, 0);
		pDC->SetWindowOrg(0,0);
		pDC->SetMapMode(MM_TEXT);
		dc.SetViewportOrg(0, 0);
		dc.SetWindowOrg(0,0);
		dc.SetMapMode(MM_TEXT);
		pDC->BitBlt(rect.left, rect.top, rect.Width(), rect.Height(),
			&dc, 0, 0, SRCCOPY);
		dc.SelectObject(pOldBitmap);
	}
}

void CPLCView::Remove(CDrawObj* pObj)
{
	if( pObj->m_drawShape == plc_start )
		return;

	POSITION pos = m_selection.Find(pObj);
	if (pos != NULL)
		m_selection.RemoveAt(pos);
}

void CPLCView::PasteNative(COleDataObject& dataObject)
{
	// get file refering to clipboard data
	CFile* pFile = dataObject.GetFileData(m_cfDraw);
	if (pFile == NULL)
		return;

	// connect the file to the archive
	CArchive ar(pFile, CArchive::load);
	TRY
	{
		ar.m_pDocument = GetDocument(); // set back-pointer in archive
		// read the selection
		m_selection.Serialize(ar);
	}
	CATCH_ALL(e)
	{
		ar.Close();
		delete pFile;
		THROW_LAST();
	}
	END_CATCH_ALL

	ar.Close();
	delete pFile;
```

```
}

void CPLCView::PasteEmbedded(COleDataObject& dataObject)
{
	BeginWaitCursor();
	// paste embedded
	CDrawOleObj* pObj = new CDrawOleObj(GetInitialPosition());
	ASSERT_VALID(pObj);
	CDrawItem* pItem = new CDrawItem(GetDocument(), pObj);
	ASSERT_VALID(pItem);
	pObj->m_pClientItem = pItem;

	TRY
	{
		if (!pItem->CreateFromData(&dataObject) &&
			!pItem->CreateStaticFromData(&dataObject))
			AfxThrowMemoryException();	// any exception will do

		// add the object to the document
		GetDocument()->Add(pObj);
		m_selection.AddTail(pObj);

		// try to get initial presentation data
		pItem->UpdateLink();
		pItem->UpdateExtent();
	}
	CATCH_ALL(e)
	{
		// clean up item
		pItem->Delete();
		pObj->m_pClientItem = NULL;
		GetDocument()->Remove(pObj);
		pObj->Remove();
		AfxMessageBox(IDP_FAILED_TO_CREATE);
	}
	END_CATCH_ALL
	EndWaitCursor();
}

void CPLCView::DrawGrid(CDC* pDC)
{
	CPLCDoc* pDoc = GetDocument();
	COLORREF oldBkColor = pDC->SetBkColor(pDoc->GetPaperColor());
	CRect rect;
	rect.left = -pDoc->GetSize().cx / 2;
	rect.top = -pDoc->GetSize().cy / 2;
	rect.right = rect.left + pDoc->GetSize().cx;
	rect.bottom = rect.top + pDoc->GetSize().cy;

	// Center lines
	CPen penDash;
	penDash.CreatePen(PS_DASH, 1, m_gridColor);
	CPen* pOldPen = pDC->SelectObject(&penDash);
```

```
	pDC->MoveTo(0, rect.top);
	pDC->LineTo(0, rect.bottom);
	pDC->MoveTo(rect.left, 0);
	pDC->LineTo(rect.right, 0);

	// Major unit lines
	CPen penDot;
	penDot.CreatePen(PS_DOT, 1, m_gridColor);
	pDC->SelectObject(&penDot);

	for (int x = rect.left / 100 * 100; x < rect.right; x += 100)
	{
		if (x != 0)
		{
			pDC->MoveTo(x, rect.top);
			pDC->LineTo(x, rect.bottom);
		}
	}

	for (int y = rect.top / 100 * 100; y < rect.bottom; y += 100)
	{
		if (y != 0)
		{
			pDC->MoveTo(rect.left, y);
			pDC->LineTo(rect.right, y);
		}
	}

	// Outlines
	CPen penSolid;
	penSolid.CreatePen(PS_SOLID, 1, m_gridColor);
	pDC->SelectObject(&penSolid);
	pDC->MoveTo(rect.left, rect.top);
	pDC->LineTo(rect.right, rect.top);
	pDC->LineTo(rect.right, rect.bottom);
	pDC->LineTo(rect.left, rect.bottom);
	pDC->LineTo(rect.left, rect.top);

	pDC->SelectObject(pOldPen);
	pDC->SetBkColor(oldBkColor);
}

void CPLCView::OnInitialUpdate()
{
	CSize size = GetDocument()->GetSize();
	CClientDC dc(NULL);
	size.cx = MulDiv(size.cx, dc.GetDeviceCaps(LOGPIXELSX), 100);
	size.cy = MulDiv(size.cy, dc.GetDeviceCaps(LOGPIXELSY), 100);
	SetScrollSizes(MM_TEXT, size);
}

void CPLCView::SetPageSize(CSize size)
{
	CClientDC dc(NULL);
```

```
	size.cx = MulDiv(size.cx, dc.GetDeviceCaps(LOGPIXELSX), 100);
	size.cy = MulDiv(size.cy, dc.GetDeviceCaps(LOGPIXELSY), 100);
	SetScrollSizes(MM_TEXT, size);
	GetDocument()->UpdateAllViews(NULL, HINT_UPDATE_WINDOW, NULL);
}

// CPLCView printing

BOOL CPLCView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CPLCView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
	CScrollView::OnBeginPrinting(pDC,pInfo);
	// check page size -- user could have gone into print setup
	// from print dialog and changed paper or orientation
	GetDocument()->ComputePageSize();
}

void CPLCView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

// OLE Client support and commands

BOOL CPLCView::IsSelected(const CObject* pDocItem) const
{
	CDrawObj* pDrawObj = (CDrawObj*)pDocItem;
	if (pDocItem->IsKindOf(RUNTIME_CLASS(CDrawItem)))
		pDrawObj = ((CDrawItem*)pDocItem)->m_pDrawObj;
	return m_selection.Find(pDrawObj) != NULL;
}

void CPLCView::OnInsertObject()
{
	// Invoke the standard Insert Object dialog box to obtain information
	//  for new CDrawItem object.
	COleInsertDialog dlg;
	if (dlg.DoModal() != IDOK)
		return;
	BeginWaitCursor();

	// First create the C++ object
	CDrawOleObj* pObj = new CDrawOleObj(GetInitialPosition());
	ASSERT_VALID(pObj);
	CDrawItem* pItem = new CDrawItem(GetDocument(), pObj);
	ASSERT_VALID(pItem);
	pObj->m_pClientItem = pItem;

	// Now create the OLE object/item
```

```
	TRY
	{
		if (!dlg.CreateItem(pObj->m_pClientItem))
			AfxThrowMemoryException();

		// add the object to the document
		GetDocument()->Add(pObj);

		// try to get initial presentation data
		pItem->UpdateLink();
		pItem->UpdateExtent();

		// if insert new object -- initially show the object
		if (dlg.GetSelectionType() == COleInsertDialog::createNewItem)
			pItem->DoVerb(OLEIVERB_SHOW, this);
	}
	CATCH_ALL(e)
	{
		// clean up item
		pItem->Delete();
		pObj->m_pClientItem = NULL;
		GetDocument()->Remove(pObj);
		pObj->Remove();

		AfxMessageBox(IDP_FAILED_TO_CREATE);
	}
	END_CATCH_ALL
	EndWaitCursor();
}

// The following command handler provides the standard keyboard
//  user interface to cancel an in-place editing session.
void CPLCView::OnCancelEdit()
{
	// deactivate any in-place active item on this view!
	COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
	if (pActiveItem != NULL)
	{
		// if we found one, deactivate it
		pActiveItem->Close();
	}
	ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);

	// escape also brings us back into select mode
	ReleaseCapture();

	CPLCDrawTool* pTool = CPLCDrawTool::FindTool(CPLCDrawTool::c_drawShape);
	if (pTool != NULL)
		pTool->OnCancel();

	CPLCDrawTool::c_drawShape = selection;
}

void CPLCView::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();
			return;
		}
	}

	CScrollView::OnSetFocus(pOldWnd);
}

CRect CPLCView::GetInitialPosition()
{
	CRect rect(10, 10, 10, 10);
	ClientToDoc(rect);
	return rect;
}

void CPLCView::ClientToDoc(CPoint& point)
{
	CClientDC dc(this);
	OnPrepareDC(&dc, NULL);
	dc.DPtoLP(&point);
}

void CPLCView::ClientToDoc(CRect& rect)
{
	CClientDC dc(this);
	OnPrepareDC(&dc, NULL);
	dc.DPtoLP(rect);
	ASSERT(rect.left <= rect.right);
	ASSERT(rect.bottom <= rect.top);
}

void CPLCView::DocToClient(CPoint& point)
{
	CClientDC dc(this);
	OnPrepareDC(&dc, NULL);
	dc.LPtoDP(&point);
}

void CPLCView::DocToClient(CRect& rect)
{
	CClientDC dc(this);
	OnPrepareDC(&dc, NULL);
	dc.LPtoDP(rect);
	rect.NormalizeRect();
}
```

```
void CPLCView::Select(CDrawObj* pObj, BOOL bAdd)
{
	if( pObj && pObj->m_bMoveable == FALSE && pObj->m_bDeletable == FALSE )
		return;
	if (!bAdd)
	{
		OnUpdate(NULL, HINT_UPDATE_SELECTION, NULL);
		m_selection.RemoveAll();
	}

	if (pObj == NULL || IsSelected(pObj))
		return;

	m_selection.AddTail(pObj);
	InvalObj(pObj);
}

// rect is in device coordinates
void CPLCView::SelectWithinRect(CRect rect, BOOL bAdd)
{
	if (!bAdd)
		Select(NULL);

	ClientToDoc(rect);

	CDrawObjList* pObList = GetDocument()->GetObjects();
	POSITION posObj = pObList->GetHeadPosition();
	while (posObj != NULL)
	{
		CDrawObj* pObj = pObList->GetNext(posObj);
		if (pObj->Intersects(rect))
			Select(pObj, TRUE);
	}
}

void CPLCView::Deselect(CDrawObj* pObj)
{
	POSITION pos = m_selection.Find(pObj);
	if (pos != NULL)
	{
		InvalObj(pObj);
		m_selection.RemoveAt(pos);
	}
}

void CPLCView::CloneSelection()
{
	POSITION pos = m_selection.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_selection.GetNext(pos);
		pObj->Clone(pObj->m_pDocument);
		// copies object and adds it to the document
```

```
        }
}

void CPLCView::UpdateActiveItem()
{
        COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
        if (pActiveItem != NULL &&
                pActiveItem->GetItemState() == COleClientItem::activeUIState)
        {
                // this will update the item rectangles by calling
                //  OnGetPosRect & OnGetClipRect.
                pActiveItem->SetItemRects();
        }
}

// CPLCView message handlers
void CPLCView::OnLButtonDown(UINT nFlags, CPoint point)
{
        if (!m_bActive)
                return;
        CPLCDrawTool* pTool = CPLCDrawTool::FindTool(CPLCDrawTool::c_drawShape);
        if (pTool != NULL)
                pTool->OnLButtonDown(this, nFlags, point);
}

void CPLCView::OnLButtonUp(UINT nFlags, CPoint point)
{
        if (!m_bActive)
                return;
        CPLCDrawTool* pTool = CPLCDrawTool::FindTool(CPLCDrawTool::c_drawShape);
        if (pTool != NULL)
                pTool->OnLButtonUp(this, nFlags, point);
}

void CPLCView::OnMouseMove(UINT nFlags, CPoint point)
{
        if (!m_bActive)
                return;
        CPLCDrawTool* pTool = CPLCDrawTool::FindTool(CPLCDrawTool::c_drawShape);
        if (pTool != NULL)
                pTool->OnMouseMove(this, nFlags, point);
}

void CPLCView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
        if (!m_bActive)
                return;
        CPLCDrawTool* pTool = CPLCDrawTool::FindTool(CPLCDrawTool::c_drawShape);
        if (pTool != NULL)
                pTool->OnLButtonDblClk(this, nFlags, point);
}

void CPLCView::OnDestroy()
{
```

```
	CScrollView::OnDestroy();

	// deactivate the inplace active item on this view
	COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
	if (pActiveItem != NULL && pActiveItem->GetActiveView() == this)
	{
		pActiveItem->Deactivate();
		ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
	}
}

void CPLCView::OnDrawSelect()
{
	CPLCDrawTool::c_drawShape = selection;
}

void CPLCView::OnDrawRoundRect()
{
	CPLCDrawTool::c_drawShape = roundRect;
}

void CPLCView::OnDrawRect()
{
	CPLCDrawTool::c_drawShape = rect;
}

BOOL CPLCView::ProcessEndInstr()
{
	if( GetDocument()->EndAdded() )
	{
		if( AfxMessageBox( ID_STR_INSERT_RUNG, MB_YESNO ) == IDYES )
		{
			int nY = GetDocument()->GetEndRungY();
			GetDocument()->SetEndRungY( nY + DEFAULT_LINE_DIST );
			InsertRung( nY );
		}
		return TRUE; // ondrawline will do nothing
	}
	return FALSE;
}

void CPLCView::InsertRung( int nY, BOOL bVerticalOnly )
{
	GetDocument()->DeleteVertical( this );

	int nLastY = GetNextHorizLineYpos();

	//draw the vertical lines for the rung of the ladder
	DrawLine( CRect( START_LINE_BEGIN_X, START_LINE_BEGIN_Y,
		START_LINE_BEGIN_X, nLastY ),
		CDrawRect::left_vertical, FALSE, FALSE );
	DrawLine( CRect( GetDocument()->LINE_END_X, START_LINE_BEGIN_Y,
		GetDocument()->LINE_END_X, nLastY ),
		CDrawRect::right_vertical, FALSE, FALSE  );
```

```
        // move all the other object down by DEFAULT_LINE_DIST
        MoveObjectsDown( GetRungPosition( nY ) );

        if( !bVerticalOnly )
        {
                //draw the horizontal line
                CDrawRect* pLine = DrawLine( CRect( START_LINE_BEGIN_X, nY,
                        GetDocument()->LINE_END_X, nY ), CDrawRect::rung, FALSE, FALSE );

                SetCurrentXY( GetCurrentLineY(), START_LINE_BEGIN_X );
                CInstructRung* pRung = GetLadder()->AddRung( pLine, TRUE );
                pLine->m_pRung = pRung;

                SetCursor( GetCurrentLineY(), START_LINE_BEGIN_X, pRung );
        }

        Invalidate();
}

POSITION CPLCView::GetRungPosition( int nY )
{
        CInstructRungList* pRungList = &GetLadder()->m_arInstructionRungs;
        POSITION  mainPos = pRungList->GetHeadPosition();
        POSITION oldPos;
        while( mainPos )
        {
                oldPos = mainPos;
                CInstructRung* pRung = pRungList->GetNext( mainPos );
                CRect rect = pRung->m_pMainRungLine->m_position;
                DocToClient( rect );
                if( rect.top >= nY )
                        return oldPos;
        }
        return NULL;
}

void CPLCView::MoveObjectsDown( POSITION rPos )
{
        if( !rPos )
        {
                ASSERT(FALSE);
                return;
        }

        CInstructRungList* pRungList = &GetLadder()->m_arInstructionRungs;
        while( rPos )
        {
                CInstructRung* pRung = pRungList->GetNext( rPos );
                CRect rect = pRung->m_pMainRungLine->m_position;
                DocToClient( rect );
                rect.top += DEFAULT_LINE_DIST;
                rect.bottom += DEFAULT_LINE_DIST;
                ClientToDoc( rect );
```

```
		pRung->m_pMainRungLine->m_position = rect;

		CObArray& rArrayInstruct = pRung->m_arInstructions;
		int nSize = rArrayInstruct.GetSize();
		for( int i = 0; i < nSize; i++ )
		{
			CInstruction* pInstruction = (CInstruction*)rArrayInstruct[i];
			if( pInstruction->m_pDrawObj )
			{
				rect = pInstruction->m_pDrawObj->m_position;
				DocToClient( rect );
				rect.top += DEFAULT_LINE_DIST;
				rect.bottom += DEFAULT_LINE_DIST;
				ClientToDoc( rect );
				if( rect.top > rect.bottom )
				{
					// swap them
					int a = rect.top;
					rect.top = rect.bottom;
					rect.bottom = a;
				}

				pInstruction->m_pDrawObj->m_position = rect;
			}
		}
	}
	IncrLastRungY( DEFAULT_LINE_DIST );
}

void CPLCView::OnDrawLine()
{
	if( ProcessEndInstr() )
		return;

	BOOL bEnd = FALSE;
	if( CPLCDrawTool::c_drawShape == plc_end )
		bEnd = TRUE;

	CPLCDrawTool::c_drawShape = line;

	int nPreviousLineY = GetLastRungY();
	int nYPos = GetNextHorizLineYpos();

	// find the size of the window
	CRect rect;
	GetWindowRect( &rect );
	CSize windowSize = rect.Size();
	if( nYPos > windowSize.cy )
	{
		CScrollBar* pScroll = GetScrollBarCtrl( SB_VERT );
		ASSERT( pScroll );
		OnVScroll(SB_LINEDOWN, 0, pScroll) ;
		OnVScroll(SB_LINEDOWN, 0, pScroll) ;
```

```
		// update the values
		nPreviousLineY = GetCurrentLineY();
		nYPos = GetMaxY();
	}

	//draw the vertical lines for the rung of the ladder
	DrawLine( CRect( START_LINE_BEGIN_X, nPreviousLineY,
		START_LINE_BEGIN_X, nYPos ), CDrawRect::left_vertical, FALSE, FALSE );
	DrawLine( CRect( GetDocument()->LINE_END_X, nPreviousLineY,
		GetDocument()->LINE_END_X, nYPos ),
		CDrawRect::right_vertical, FALSE, FALSE  );

	//draw the  line
	CDrawRect* pLine = DrawLine( CRect( START_LINE_BEGIN_X - OVERLAP_WIDTH,
		nYPos, GetDocument()->LINE_END_X, nYPos ),
		bEnd ? CDrawRect::end : CDrawRect: rung, FALSE, FALSE );
	SetCurrentXY( GetCurrentLineY(), START_LINE_BEGIN_X );
	CInstructRung* pRung = GetLadder()->AddRung( pLine );
	pLine->m_pRung = pRung;
	SetLastRungY( GetCurrentLineY() );

	SetCursor( GetCurrentLineY(), START_LINE_BEGIN_X, pRung );
	OnDrawSelect();
}

void CPLCView::OnDrawEllipse()
{
	CPLCDrawTool::c_drawShape = ellipse;
}

void CPLCView::OnDrawPolygon()
{
	CPLCDrawTool::c_drawShape = poly;
}

BOOL CPLCView::CheckCursor()
{
	if( GetDocument()->m_cursor.m_pCurrentLine == NULL )
	{
		MessageBeep(MB_ICONHAND);
		AfxMessageBox(IDS_ILLEGAL_CURSOR_POSITION );
		return FALSE;
	}
	return TRUE;
}

BOOL CPLCView::OutputSwitchPresent()
{
	CDrawRect* pLine = GetCurrentLine();

	if( pLine->m_LineType == CDrawRect::right_vertical ||
		pLine->m_LineType == CDrawRect::left_vertical )
	{
		MessageBeep(MB_ICONHAND);
```

```
		((CMainFrame*)AfxGetApp()->m_pMainWnd)->
			SetStatusMessage( IDS_ILLEGAL_CURSOR_POSITION );
		return FALSE;
	}

	CInstructRung* pRung = pLine->m_pRung;
	ASSERT( pRung );

	if( pRung->GetOutputInstruction())
	{
		MessageBeep(MB_ICONHAND);
		AfxMessageBox(IDS_ILLEGAL_SWITCH_POSITION );
		return TRUE;
	}
	return FALSE;
}

// If EndRungPresent placement of switch is not allowed.
BOOL CPLCView::EndRungPresent()
{
	if(GetDocument()->m_cursor.m_pCurrentLine->m_LineType == CDrawRect::end )
	{
		MessageBeep(MB_ICONHAND);
		AfxMessageBox(IDS_ILLEGAL_SWITCH_POSITION_FORENDRUNG );
		return TRUE;
	}
	return FALSE;
}

void CPLCView::OnDrawSerialOpenSwitch()
{
	if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
	CPLCDrawTool::c_drawShape = serial_open;
	if( !DrawComponent() ) return;
}

void CPLCView::OnDrawSerialClosedSwitch()
{
	if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
	CPLCDrawTool::c_drawShape = serial_closed;
	if( !DrawComponent() ) return;
}

void CPLCView::OnDrawOutput()
{
	if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
	CInstructRung* pRung = GetCurrentLine()->m_pRung;
	ASSERT(pRung);
	if( !pRung )
		return;

	if( !pRung->m_nCurrentInstruct )
	{
		MessageBeep(MB_ICONHAND);
```

```
                AfxMessageBox( IDS_ILLEGAL_OUTPUT_POSITION );
                return;
        }

        CPLCDrawTool::c_drawShape = plc_output;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawSetSwitch()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = set_switch;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawResetSwitch()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = reset_switch;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawMove()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = plc_move;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawSubtraction()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = minus;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawAddition()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = plus;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawEqualto()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = equal_to;
        if( !DrawComponent() ) return;
}

void CPLCView::OnDrawGreaterthan()
{
        if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = greater_than;
```

```
	if( !DrawComponent() ) return;
}

void CPLCView::OnDrawLessthan()
{
	if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
	CPLCDrawTool::c_drawShape = less_than;
	if( !DrawComponent() ) return;
}

void CPLCView::OnDrawStart()
{
	CPLCDrawTool::c_drawShape = plc_start;
	if( !DrawComponent() ) return;

	//draw the horizontal line before and after the start

	int nHalfX = (START_LINE_BEGIN_X + GetDocument()->LINE_END_X)/2;

	int nStartWidth = plcStart.m_nBitmapWidth;

	DrawLine( CRect( START_LINE_BEGIN_X - OVERLAP_WIDTH, START_LINE_BEGIN_Y,
		nHalfX - nStartWidth/2, START_LINE_BEGIN_Y ),
		CDrawRect::start, FALSE, FALSE  );

	GetDocument()->m_startLine = DrawLine( CRect( nHalfX + nStartWidth/2 -
		OVERLAP_WIDTH, START_LINE_BEGIN_Y,
	GetDocument()->LINE_END_X, START_LINE_BEGIN_Y ),
		CDrawRect::start, FALSE, FALSE  );
	// create the first rung by default
	OnDrawLine();
}


void CPLCView::OnDrawStop()
{
	//draw the ladder rung
	CPLCDrawTool::c_drawShape = plc_end;
	OnDrawLine();

	CPLCDrawTool::c_drawShape = plc_end;
	if( !DrawComponent() ) return;
}

void CPLCView::OnDrawTimer()
{
	if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
	CPLCDrawTool::c_drawShape = plc_timer;
	if( !DrawComponent() ) return;
}

void CPLCView::OnDrawCounter()
{
	if( !CheckCursor() || OutputSwitchPresent() || EndRungPresent()) return;
```

```
        CPLCDrawTool::c_drawShape = plc_counter;
        if( !DrawComponent() ) return;
}

BOOL CPLCView::MakeSpace( int nY, CInstructRung* pCurrentRung )
{
        // don't let this shunt be drawn over any existing rung
        // find any rung which is within 25 pixel of the given nY

        CInstructRung* pRung;
        pRung = GetLadder()->GetNearestRungBelow( nY, this,pCurrentRung );

        if( pRung )
        {
                // there is a rung very close to this shunt, move it below.
                InsertRung( nY, TRUE );
                return TRUE;
        }
        return FALSE;
}

void CPLCView::OnDrawShunt()
{
        if (EndRungPresent()) return;
        CPLCDrawTool::c_drawShape = shunt;

        int nY = GetCurrentLineY();
        int nLine = GetCurrentLineNumber();
        CInstructRung* pRung = GetCurrentLine()->m_pRung;

        int pos;
        if( !pRung || !pRung->InstructionAtRight( this, &pos ) )
        {
                ((CMainFrame*)AfxGetApp()->m_pMainWnd)->
                        SetStatusMessage( IDS_STRING_NO_COMPONENT );
                MessageBeep(MB_ICONHAND);
                CPLCDrawTool::c_drawShape = selection;
                return;
        }

        // don't let this shunt be drawn over any existing rung
        BOOL bInserted = MakeSpace( nY + DEFAULT_LINE_DIST, pRung );

        int nX = pRung->m_nCurrentX;
        CDrawRect* pLine1 = DrawLine( CRect( nX, nY, pRung->m_nCurrentX, nY +
                DEFAULT_LINE_DIST ), CDrawRect::shunt, FALSE, FALSE  );

        // draw a small horizontal line.
        nX += 10;
        CDrawRect* pLine2 = DrawLine( CRect( nX-10, nY + DEFAULT_LINE_DIST,
                nX, nY + DEFAULT_LINE_DIST ), CDrawRect::shunt, FALSE, FALSE  );

        //update the current line #
        SetCurrentLine( pLine2 );
```

```
	CInstructRung* pChildRung;
	// mark the beginning of parallel rung in the main rung
	pChildRung = pRung->AddParallelRung( GetLadder()->
		GetNextRungNumber(), nX, pos, pLine1, pLine2 );
	pChildRung->m_pParentRung = pRung;
	pLine1->m_pRung = pChildRung;
	pLine2->m_pRung = pChildRung;

	if( !bInserted )	// update the YMax
		GetNextHorizLineYpos();

	SetCurrentXY( nY+ DEFAULT_LINE_DIST, pChildRung->m_nCurrentX );
	SetCursor( GetCurrentLineY(), pChildRung->m_nCurrentX, pChildRung );
	OnDrawSelect();
	m_selectMode = parallel_rung;
}

void CPLCView::OnDrawUpwardShunt()
{
	m_selectMode = complete_rung;
	((CMainFrame*)AfxGetApp()->m_pMainWnd)->
		SetStatusMessage( IDS_STRING_COMPLETE_SHUNT );
}


BOOL CPLCView::DrawComponent( DrawShape shape )
{
	DrawShape useShape;
	if( shape == selection )
		useShape = CPLCDrawTool::c_drawShape;
	else
		useShape = shape;

	CInstructRung* pRung;
	CDrawRect* pLine = GetCurrentLine();
	if( useShape != plc_start )// && useShape != plc_end )
	{
		pRung = pLine->m_pRung;
		if( !pRung )
		{
			MessageBeep(MB_ICONHAND);
			((CMainFrame*)AfxGetApp()->m_pMainWnd)->
				SetStatusMessage( IDS_BAD_CURSOR_POS );
			CPLCDrawTool::c_drawShape = selection;
			return FALSE;
		}
	}

	CPLCComponentDrawTool* pTool = CDrawObj::GetTool( useShape );
	CString strValue1, strValue2;
	if( pTool->m_nInputs )
	{
		// show the dialog to get the input/output card number
```

```
		CSetObjNameDlg dlg( pTool, NULL, this );
		if( dlg.DoModal() != IDOK )
		{
			// reset selection
			CPLCDrawTool::c_drawShape = selection;
			return FALSE;
		}

		strValue1 = dlg.GetValue1();

		if( pTool->m_nInputs == 2 )
			strValue2 = dlg.GetValue2();
	}

	CPoint local;
	BOOL bDeleteable = TRUE;
	BOOL bMoveable = TRUE;

	switch( useShape )
	{
		case plc_start :
			local.x = (START_LINE_BEGIN_X + GetDocument()->
				LINE_END_X)/2 - plcStart.m_nBitmapWidth/2;
			local.y = START_LINE_BEGIN_Y + pTool->m_nConnectorHeight;
			//bDeleteable = FALSE;
			//bMoveable = FALSE;
			break;

		default :
			// we have to find the current line height and the X value to place the object
			local.y = GetCurrentLineY() + pTool->m_nConnectorHeight;

			if( useShape == plc_end )
			{
				//bDeleteable = FALSE;
				local.x = (START_LINE_BEGIN_X +
					GetDocument()->LINE_END_X)/2;
			}
			else
				local.x = pRung->m_nNextObjectX;

			pRung->m_nNextObjectX += pTool->m_nBitmapWidth;

			if( pRung->m_nNextObjectX +
				INTER_COMPONENT_SPACE > GetDocument()->LINE_END_X )
			{
				// delete the right vertical line
				// increase x of all rungs
				GetDocument()->MoveRightLine( this );

				// time to increase the ladder width
				GetDocument()->LINE_END_X += DEFAULT_X_INCREMENT ;

				// redraw the right vertical line
```

```
                        CRect topRect = GetDocument()->m_startLine->m_position;
                        DocToClient( topRect );
                        CRect rect( GetDocument()->LINE_END_X , topRect.top,
                                GetDocument()->LINE_END_X , topRect.top +
                                GetDocument()->m_cursor.m_nAbsoluteLastY );

                        DrawLine( rect, CDrawRect::right_vertical, FALSE, FALSE  );

                        CRect rectCurLine = pRung->m_pMainRungLine->m_position;
                        DocToClient( rectCurLine );
                        SetCurrentXY( rectCurLine.bottom, pRung->m_nNextObjectX );
                }

                if( !pRung->m_bChildRung )
                        pRung->m_nNextObjectX += INTER_COMPONENT_SPACE;
                  pRung->m_nCurrentX = pRung->m_nNextObjectX;

                //change m_nNextObjectX so as to place the cursor just for the line end
                SetCurrentXY( GetCurrentLineY(), pRung->m_nNextObjectX );
                SetCursor( GetCurrentLineY(), pRung->m_nNextObjectX , pRung );
        }

        ClientToDoc(local);
        CRect rect( local, CSize( pTool->m_nBitmapWidth, pTool->m_nBitmapHeight ) );
        CDrawBitmap* pObj = new CDrawBitmap( rect, useShape,
                bMoveable, bDeleteable, strValue1, strValue2 );

        if( useShape != plc_start )
                pObj->m_pLine = GetCurrentLine();

        //update the instructions generated
        int nParam1, nParam2;
        if( pTool->m_nHexChars )
        {
                char* pszEnd;
                nParam1 = strtol( strValue1, &pszEnd, 16 );
                nParam2 = strtol( strValue2, &pszEnd, 16 );
        }
        else
        {
                nParam1 = atoi( strValue1 );
                nParam2 = atoi( strValue2 );
        }

        if( useShape != plc_start )
                pRung->AddInstruction( pObj );

        GetDocument()->Add(pObj);
        OnDrawSelect();
        m_nDragHandle = 1;
        m_lastPoint = local;
        Invalidate();
        return TRUE;
}
```

```
void CPLCView::OnUpdateUpwardShunt(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded || m_selectMode != parallel_rung )
	{
		pCmdUI->Enable(FALSE);
	}
	else
	if( m_selectMode == parallel_rung )
	{
		CInstructRung* pRung = GetCurrentLine()->m_pRung;
		if( pRung && pRung->m_nCurrentInstruct )
			pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == upward_shunt);
		else
			pCmdUI->Enable(FALSE);
	}
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == upward_shunt);
}

void CPLCView::OnUpdateDrawShunt(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == shunt);
}

void CPLCView::OnUpdateDrawEllipse(CCmdUI* pCmdUI)
{
	pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == ellipse);
}

void CPLCView::OnUpdateDrawLine(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == serial_closed);
}

void CPLCView::OnUpdateDrawRect(CCmdUI* pCmdUI)
{
	pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == rect);
}

void CPLCView::OnUpdateDrawRoundRect(CCmdUI* pCmdUI)
{
	pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == roundRect);
}

void CPLCView::OnUpdateSerialOpen(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
```

```
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == serial_open);
}

void CPLCView::OnUpdateDrawSerialClosedSwitch(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == serial_closed);
}

void CPLCView::OnUpdateDrawOutput(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plc_output);
}

void CPLCView::OnUpdateDrawResetSwitch(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == reset_switch);
}

void CPLCView::OnUpdateDrawSetSwitch(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == set_switch);
}

void CPLCView::OnUpdateDrawMove(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plc_move);
}

void CPLCView::OnUpdateDrawSubtraction(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == minus);
}

void CPLCView::OnUpdateDrawAddition(CCmdUI* pCmdUI)
```

```
{
        if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plus);
}

void CPLCView::OnUpdateDrawEqualto(CCmdUI* pCmdUI)
{
        if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == equal_to);
}

void CPLCView::OnUpdateDrawGreaterthan(CCmdUI* pCmdUI)
{
        if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == greater_than);
}

void CPLCView::OnUpdateDrawLessthan(CCmdUI* pCmdUI)
{
        if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == less_than);
}

void CPLCView::OnUpdateDrawSelect(CCmdUI* pCmdUI)
{
        if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == selection);
}

void CPLCView::OnUpdateSingleSelect(CCmdUI* pCmdUI)
{
        if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->Enable(m_selection.GetCount() == 1);
}

void CPLCView::OnUpdateDrawStart(CCmdUI* pCmdUI)
{
        if( ((CPLCDoc*)GetDocument())->m_bStartAdded )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plc_start);
}
```

```
void CPLCView::OnUpdateDrawStop(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plc_end);
}

void CPLCView::OnUpdateDrawTimer(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plc_timer);
}

void CPLCView::OnUpdateDrawCounter(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == plc_counter );
}

CDrawRect* CPLCView::DrawLine( CRect& r, CDrawRect::LineType type, BOOL bMoveable, BOOL
bDeleteable )
{
	CRectTool tool(line);
	CPoint linePoint( r.left, r.top );

	tool.OnLButtonDown(this, 0, linePoint);

	linePoint = CPoint( r.right, r.bottom );

	tool.OnMouseMove(this, 0, linePoint);
	tool.OnLButtonUp(this, 0, linePoint);

	// if it is a horizontal line, set current y to top, else for vertical line to bottom
	if( r.bottom > r.top )
		SetCurrentLineY( r.bottom );
	else
		SetCurrentLineY( r.top );

	//find the line just added
	CDrawRect* pObj = (CDrawRect*)GetDocument()->GetLastObject();
	if( pObj )
	{
		pObj->m_bMoveable = bMoveable;
		pObj->m_bDeletable = bDeleteable;
		pObj->m_LineType = type;
		return pObj;
	}
	return NULL;
```

```
}

void CPLCView::OnEditSelectAll()
{
	CDrawObjList* pObList = GetDocument()->GetObjects();
	POSITION pos = pObList->GetHeadPosition();
	while (pos != NULL)
		Select(pObList->GetNext(pos), TRUE);
}

void CPLCView::OnUpdateEditSelectAll(CCmdUI* pCmdUI)
{
	pCmdUI->Enable(GetDocument()->GetObjects()->GetCount() != 0);
}

void CPLCView::OnEditClear()
{
	// update all the views before the selection goes away
	GetDocument()->UpdateAllViews(NULL, HINT_DELETE_SELECTION, &m_selection);
	OnUpdate(NULL, HINT_UPDATE_SELECTION, NULL);

	// now remove the selection from the document
	POSITION pos = m_selection.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_selection.GetNext(pos);

		if( pObj->m_drawShape == plc_start )
			continue;

		((CDrawBitmap*)pObj)->m_pLine->m_pRung->Delete( pObj );

		GetDocument()->Remove(pObj);
		pObj->Remove();
	}
	m_selection.RemoveAll();
}

void CPLCView::OnUpdateAnySelect(CCmdUI* pCmdUI)
{
	pCmdUI->Enable(!m_selection.IsEmpty());
}

void CPLCView::OnUpdateDrawPolygon(CCmdUI* pCmdUI)
{
	if( !((CPLCDoc*)GetDocument())->m_bStartAdded )
		pCmdUI->Enable(FALSE);
	else
		pCmdUI->SetRadio(CPLCDrawTool::c_drawShape == poly);
}

void CPLCView::OnSize(UINT nType, int cx, int cy)
{
	CScrollView::OnSize(nType, cx, cy);
```

```
	UpdateActiveItem();
}

void CPLCView::OnViewGrid()
{
	m_bGrid = !m_bGrid;
	Invalidate(FALSE);
}

void CPLCView::OnUpdateViewGrid(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(m_bGrid);
}

BOOL CPLCView::OnEraseBkgnd(CDC*)
{
	return TRUE;
}

void CPLCView::OnObjectFillColor()
{
	CColorDialog dlg;
	if (dlg.DoModal() != IDOK)
		return;

	COLORREF color = dlg.GetColor();

	POSITION pos = m_selection.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_selection.GetNext(pos);
		pObj->SetFillColor(color);
	}
}

void CPLCView::OnObjectLineColor()
{
	CColorDialog dlg;
	if (dlg.DoModal() != IDOK)
		return;

	COLORREF color = dlg.GetColor();

	POSITION pos = m_selection.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_selection.GetNext(pos);
		pObj->SetLineColor(color);
	}
}

void CPLCView::OnObjectMoveBack()
{
	CPLCDoc* pDoc = GetDocument();
```

```
        CDrawObj* pObj = m_selection.GetHead();
        CDrawObjList* pObjects = pDoc->GetObjects();
        POSITION pos = pObjects->Find(pObj);
        ASSERT(pos != NULL);
        if (pos != pObjects->GetHeadPosition())
        {
                POSITION posPrev = pos;
                pObjects->GetPrev(posPrev);
                pObjects->RemoveAt(pos);
                pObjects->InsertBefore(posPrev, pObj);
                InvalObj(pObj);
        }
}

void CPLCView::OnObjectMoveForward()
{
        CPLCDoc* pDoc = GetDocument();
        CDrawObj* pObj = m_selection.GetHead();
        CDrawObjList* pObjects = pDoc->GetObjects();
        POSITION pos = pObjects->Find(pObj);
        ASSERT(pos != NULL);
        if (pos != pObjects->GetTailPosition())
        {
                POSITION posNext = pos;
                pObjects->GetNext(posNext);
                pObjects->RemoveAt(pos);
                pObjects->InsertAfter(posNext, pObj);
                InvalObj(pObj);
        }
}

void CPLCView::OnObjectMoveToBack()
{
        CPLCDoc* pDoc = GetDocument();
        CDrawObj* pObj = m_selection.GetHead();
        CDrawObjList* pObjects = pDoc->GetObjects();
        POSITION pos = pObjects->Find(pObj);
        ASSERT(pos != NULL);
        pObjects->RemoveAt(pos);
        pObjects->AddHead(pObj);
        InvalObj(pObj);
}

void CPLCView::OnObjectMoveToFront()
{
        CPLCDoc* pDoc = GetDocument();
        CDrawObj* pObj = m_selection.GetHead();
        CDrawObjList* pObjects = pDoc->GetObjects();
        POSITION pos = pObjects->Find(pObj);
        ASSERT(pos != NULL);
        pObjects->RemoveAt(pos);
        pObjects->AddTail(pObj);
        InvalObj(pObj);
}
```

```
void CPLCView::OnEditCopy()
{
	ASSERT_VALID(this);
	ASSERT(m_cfDraw != NULL);

	// Create a shared file and associate a CArchive with it
	CSharedFile file;
	CArchive ar(&file, CArchive::store);

	// Serialize selected objects to the archive
	m_selection.Serialize(ar);
	ar.Close();

	COleDataSource* pDataSource = NULL;
	TRY
	{
		pDataSource = new COleDataSource;
		// put on local format instead of or in addation to
		pDataSource->CacheGlobalData(m_cfDraw, file.Detach());

		// if only one item and it is a COleClientItem then also
		// paste in that format
		CDrawObj* pDrawObj = m_selection.GetHead();
		if (m_selection.GetCount() == 1 &&
			pDrawObj->IsKindOf(RUNTIME_CLASS(CDrawOleObj)))
		{
			CDrawOleObj* pDrawOle = (CDrawOleObj*)pDrawObj;
			pDrawOle->m_pClientItem->GetClipboardData(pDataSource, FALSE);
		}
		pDataSource->SetClipboard();
	}
	CATCH_ALL(e)
	{
		delete pDataSource;
		THROW_LAST();
	}
	END_CATCH_ALL
}

void CPLCView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
	pCmdUI->Enable(!m_selection.IsEmpty());
}

void CPLCView::OnEditCut()
{
	OnEditCopy();
	OnEditClear();
}

void CPLCView::OnUpdateEditCut(CCmdUI* pCmdUI)
{
	pCmdUI->Enable(!m_selection.IsEmpty());
```

```
}

void CPLCView::OnEditPaste()
{
	COleDataObject dataObject;
	dataObject.AttachClipboard();

	// invalidate current selection since it will be deselected
	OnUpdate(NULL, HINT_UPDATE_SELECTION, NULL);
	m_selection.RemoveAll();
	if (dataObject.IsDataAvailable(m_cfDraw))
	{
		PasteNative(dataObject);

		// now add all items in m_selection to document
		POSITION pos = m_selection.GetHeadPosition();
		while (pos != NULL)
			GetDocument()->Add(m_selection.GetNext(pos));
	}
	else
		PasteEmbedded(dataObject);

	GetDocument()->SetModifiedFlag();

	// invalidate new pasted stuff
	GetDocument()->UpdateAllViews(NULL, HINT_UPDATE_SELECTION, &m_selection);
}

void CPLCView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
	// determine if private or standard OLE formats are on the clipboard
	COleDataObject dataObject;
	BOOL bEnable = dataObject.AttachClipboard() &&  (dataObject.IsDataAvailable(m_cfDraw) ||
		COleClientItem::CanCreateFromData(&dataObject));

	// enable command based on availability
	pCmdUI->Enable(bEnable);
}

void CPLCView::OnFilePrint()
{
	CScrollView::OnFilePrint();
	GetDocument()->ComputePageSize();
}

void CPLCView::OnViewShowObjects()
{
	CDrawOleObj::c_bShowItems = !CDrawOleObj::c_bShowItems;
	GetDocument()->UpdateAllViews(NULL, HINT_UPDATE_OLE_ITEMS, NULL);
}

void CPLCView::OnUpdateViewShowObjects(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(CDrawOleObj::c_bShowItems);
```

```
}

void CPLCView::OnEditProperties()
{
	if (m_selection.GetCount() == 1 && CPLCDrawTool::c_drawShape == selection)
	{
		CPLCDrawTool* pTool = CPLCDrawTool::FindTool(CPLCDrawTool::c_drawShape);
		ASSERT(pTool != NULL);
		pTool->OnLButtonDblClk(this, 0, CPoint(0, 0));
	}
}

void CPLCView::OnUpdateEditProperties(CCmdUI* pCmdUI)
{
	pCmdUI->Enable(m_selection.GetCount() == 1 &&
		CPLCDrawTool::c_drawShape == selection);
}

// CPLCView diagnostics

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

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

//Get or Set or modify the members  of the Document
int CPLCView::GetMaxY()
{
	return GetDocument()->m_nMaxY;
}

int CPLCView::GetNextHorizLineYpos()
{
	GetDocument()->m_nMaxY += DEFAULT_LINE_DIST;
	return GetDocument()->m_nMaxY;
}

CLadder* CPLCView::GetLadder()
{
	return GetDocument()->GetLadder();
}

int CPLCView::GetNextLineNumber()
{
	return GetDocument()->m_cursor.m_nNextLineNumber;
}
```

```
int CPLCView::IncrNextLineNumber()
{
	return GetDocument()->m_cursor.m_nNextLineNumber++;
}

int CPLCView::GetCurrentLineNumber()
{
	return GetDocument()->m_cursor.m_nCurrentLineNum;
}

CDrawRect* CPLCView::GetCurrentLine()
{
	return GetDocument()->m_cursor.m_pCurrentLine;
}

void CPLCView::SetCurrentLine( CDrawRect* pLine )
{
	GetDocument()->m_cursor.m_pCurrentLine = pLine;
	SetCurrentLineNumber( pLine->m_nLineNumber );
}

void CPLCView::SetCurrentLineNumber( int n )
{
	GetDocument()->m_cursor.m_nCurrentLineNum = n;
}

void CPLCView::IncrLastRungY( int nY )
{
	GetDocument()->m_cursor.m_nLastRungY += nY;
	GetDocument()->m_cursor.m_nAbsoluteLastY += nY;
}

void CPLCView::SetLastRungY( int nY )
{
	GetDocument()->m_cursor.m_nLastRungY = nY;

	GetDocument()->m_cursor.m_nAbsoluteLastY += DEFAULT_LINE_DIST;
}

int CPLCView::GetLastRungY( )
{
	return GetDocument()->m_cursor.m_nLastRungY;
}

void CPLCView::IncrCurrentLineNumber()
{
	GetDocument()->m_cursor.m_nCurrentLineNum++;
}

void CPLCView::SetCurrentLineY(int nNewY)
{
	GetDocument()->m_cursor.m_nCurrentLineY = nNewY;
}
```

```
void CPLCView::ChangeCurrentLineYBy(int nSize )
{
	GetDocument()->m_cursor.m_nCurrentLineY += nSize;
}

void CPLCView::SetCurrentLineX(int nNewX)
{
	GetDocument()->m_cursor.m_nCurrentLineX = nNewX;
}

CDrawRect* CPLCView::GetPCursor()
{
	return GetDocument()->m_cursor.m_pCursor;
}

void CPLCView::SetPCursor( CDrawRect* p )
{
	GetDocument()->m_cursor.m_pCursor = p;
}

int CPLCView::GetCurrentLineY()
{
	return GetDocument()->m_cursor.m_nCurrentLineY;
}

void CPLCView::ResetCursor()
{
	GetDocument()->m_cursor.m_pCurrentLine = NULL;
}

CString CPLCView::GetCodeStr( CInstruction* pInstruction )
{
	CString strInstr, strParam2;
	strInstr.Format( "%s\t%X\t%s",  pInstruction->m_pOpCode->szMnemonic, pInstruction
		->m_pOpCode->nOpCode, (LPCSTR)pInstruction->m_pDrawObj->m_strName );

	if( !pInstruction->m_pDrawObj->m_strValue2.IsEmpty() )
	{
		strParam2.Format( "\t%s", (LPCSTR)pInstruction->m_pDrawObj->m_strValue2 );
		strInstr += strParam2;
	}
	strInstr += "\r\n";
	return strInstr;
}

void CPLCView::OnPlcGeneratecode()
{
	if( !m_pInstructDoc )
	{
		m_pInstructDoc = (CInstructDoc*)((CPLCApp*)AfxGetApp())->
			m_pInstrDocTemplate->OpenDocumentFile(NULL);
		CString strTitle = GetDocument()->GetTitle();
		strTitle += " PLC Assembly Code";
		m_pInstructDoc ->SetTitle( strTitle );
```

```
		m_pInstructDoc ->SetViewPtr( this );
	}
	else
		m_pInstructDoc->CutAll();

	m_pInstructDoc ->AppendToViews( "Mnemonic : OpCode Param1	Param2\r\n" );

	CInstructRungList* pRungList = &GetLadder()->m_arInstructionRungs;
	POSITION  mainPos = pRungList->GetHeadPosition();
	while( mainPos )
	{
		CInstructRung* pRung = pRungList->GetNext( mainPos );
		ShowGeneratedCode( pRung, TRUE, 0 );
	}
}

void CPLCView::ShowGeneratedCode( CInstructRung* pRung, BOOL bShow,
	int nParallelRungNumber )
{
	static CMapWordToOb m_hashParallelRungs;
	static int nStoreAddr = 0;
	CString* pInstructString = NULL;

	CObArray& rArrayInstruct = pRung->m_arInstructions;
	int i;

	for( i = 0; i < rArrayInstruct.GetSize(); i++ )
	{
		CInstruction* pInstruction = (CInstruction*)rArrayInstruct[i];
		if( pInstruction->m_pDrawObj )
		{
			if( bShow )
			{
				m_pInstructDoc ->AppendToViews( GetCodeStr( pInstruction ) );
			}
			else
			{
				// put in the string for the first time, create the string, put in hashtable
				if( !pInstructString )
				{
					pInstructString = new CString;
					m_hashParallelRungs.SetAt(
						nParallelRungNumber, (CObject*)pInstructString );
				}

				*pInstructString += GetCodeStr( pInstruction );
			}
		}
		else
		{
			// process parallel rung
			if( pInstruction->m_Type == CInstruction::ParallelBegin )
			{
				ShowGeneratedCode( pInstruction->m_pChildRung,
```

```
                        FALSE, pInstruction->m_nComponent );
                }
                else
                {
                        ASSERT( pInstruction->m_Type == CInstruction::ParallelEnd );
                        // find the rung #, get the instructions, show them, delete the hash table
                        CString* pStr;
                        BOOL bRet = m_hashParallelRungs.Lookup(
                                pInstruction->m_nComponent, (CObject*&)pStr );
                        ASSERT( bRet );
                        if( bRet )
                        {
                                CString strStore;
                                OpCode* pOpCode = CInstruction::GetSTA();
                                strStore.Format( "%s\t%X\t%d\r\n", pOpCode->szMnemonic,
                                        pOpCode->nOpCode, nStoreAddr );

                                if( bShow )
                                        m_pInstructDoc ->AppendToViews( strStore );
                                else
                                        *pInstructString += strStore;

                                if( bShow )
                                        m_pInstructDoc ->AppendToViews( *pStr );
                                else
                                        *pInstructString += *pStr;

                                pOpCode = CInstruction::GetORS();
                                strStore.Format( "%s\t%X\t%d\r\n", pOpCode->szMnemonic,
                                        pOpCode->nOpCode, nStoreAddr );

                                if( bShow )
                                        m_pInstructDoc ->AppendToViews( strStore );
                                else
                                        *pInstructString += strStore;

                                pOpCode = CInstruction::GetLDS();
                                strStore.Format( "%s\t%X\t%d\r\n", pOpCode->szMnemonic,
                                        pOpCode->nOpCode, nStoreAddr );

                                if( bShow )
                                        m_pInstructDoc ->AppendToViews( strStore );
                                else
                                        *pInstructString += strStore;

                                nStoreAddr++;
                                delete pStr;
                        }
                }
        }
    }
}

void CPLCView::RemoveRungFromScreen()
```

```
{
	// add all objects on the rung and its parallel rungs on m_selection
	GetDocument()->UpdateAllViews(NULL, HINT_DELETE_SELECTION, &m_selection);
	OnUpdate(NULL, HINT_UPDATE_SELECTION, NULL);
}

void CPLCView::SelectRung( CInstructRung* pRung )
{
	CObArray& rArray = pRung->m_arInstructions;
	int nSize = rArray.GetSize();

	for( int i = 0; i < nSize; i++ )
	{
		CInstruction* pInstruct = (CInstruction*)rArray[i];

		if( pInstruct->m_pDrawObj )
			m_selection.AddTail( pInstruct->m_pDrawObj );

		if( pInstruct->m_Type == CInstruction::ParallelBegin )
			SelectRung( pInstruct->m_pChildRung );
	}

	// select the horizontal and vertical line objects to delete
	m_selection.AddTail( pRung->m_pMainRungLine );
	if( pRung->m_bChildRung )
	{
		m_selection.AddTail( pRung->m_pParallelRungLeftLine );
		// this will allow half rung to be deleted
		if( pRung->m_pParallelRungRightLine )
			m_selection.AddTail( pRung->m_pParallelRungRightLine );
	}
}

void CPLCView::OnEditDeleterung()
{
	CDrawRect* pLine = GetCurrentLine();
	CInstructRung* pRung = pLine->m_pRung;
	if( pRung->m_bChildRung )
	{
		// set the cursor on the parent rung
		CRect rect = pRung->m_pParentRung->m_pMainRungLine->m_position;
		DocToClient(rect);
		SetCurrentXY( rect.bottom, pRung->m_pParentRung->m_nCurrentX );
		SetCursor( GetCurrentLineY(), pRung->m_pParentRung->m_nCurrentX,
			pRung->m_pParentRung );
	}

	SelectRung( pRung );
	RemoveRungFromScreen();
	// now remove the rung from the document
	POSITION pos = m_selection.GetHeadPosition();
	while (pos != NULL)
	{
		CDrawObj* pObj = m_selection.GetNext(pos);
```

```
		GetDocument()->Remove( pObj, TRUE );
		pObj->Remove(); // deletes pObj;
	}

	m_selection.RemoveAll();
	GetLadder()->DeleteRung( pRung );
	if( m_selectMode == parallel_rung )
		m_selectMode = none;

	ResetCursor();
}

void CPLCView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
	// print the header
	CString strName = GetDocument()->GetTitle();
	pDC->TextOut( 20, 20, strName, strName.GetLength() );

	CView::OnPrint( pDC, pInfo );
}

void CPLCView::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
	int nMapMode;
	SIZE sizeTotal, sizePage, sizeLine;
	GetDeviceScrollSizes( nMapMode, sizeTotal, sizePage, sizeLine );

	switch( nSBCode )
	{
		case SB_BOTTOM   : break;
		case SB_ENDSCROLL : break;
		case SB_LINEDOWN  :
			GetDocument()->ChangeMaxYBy( -sizeLine.cy );
			ChangeCurrentLineYBy( -sizeLine.cy );
			break;
		case SB_PAGEUP   :
			GetDocument()->ChangeMaxYBy( sizePage.cy );
			ChangeCurrentLineYBy( sizePage.cy );
			break;
		case SB_LINEUP   :
			GetDocument()->ChangeMaxYBy( sizeLine.cy );
			ChangeCurrentLineYBy( sizeLine.cy );
			break;
		case SB_PAGEDOWN  :
			GetDocument()->ChangeMaxYBy( -sizePage.cy );
			ChangeCurrentLineYBy( -sizePage.cy );
			break;

		case SB_THUMBPOSITION : return;
		case SB_THUMBTRACK    : return;
		case SB_TOP   : break;
	}

	CScrollView::OnVScroll(nSBCode, nPos, pScrollBar);
```

```
}
```

// Instruct.h

```
// instruct.h - interface for generating instructions
#ifndef __INSTRUCT_H__
#define __INSTRUCT_H__
#include "plctool.h"
#include "plcobj.h"

enum DrawShape;
class CPLCView;

struct OpCode
{
	DrawShape drawShape;
	char szMnemonic[5];
	int	nOpCode;
};

extern OpCode PLC_OpCodes[];
class CInstructRung;
class CInstruction : public CObject
{
	DECLARE_SERIAL( CInstruction );
	public :
		CInstruction(); // default constructor
		CInstruction( CDrawObj* pObj, int nComponentNumber );
		CInstruction( BOOL bParallelRungBegin, int nRungNumber,
			CInstructRung* pChild );
		~CInstruction(){}
		virtual void Serialize(CArchive& ar);

		// Code generation methods
		OpCode* GetOpCode( );
		static OpCode* GetSTA(){ return	&PLC_OpCodes[19]; }
		static OpCode* GetORS(){ return	&PLC_OpCodes[20]; }
		static OpCode* GetLDS(){ return	&PLC_OpCodes[21]; }

		DrawShape GetDrawShape(){ if( m_pDrawObj )
			return m_pDrawObj->m_drawShape; return shunt; }
		enum InstructionType{ Instruction, ParallelBegin, ParallelEnd };

		// normal instruction or placeholder for parallel rung
		InstructionType m_Type;
		// used for parallel child rungs
		CInstructRung* m_pChildRung;
		// component # or parallel rung's parent rung #
		int m_nComponent;
		int m_nRung;				// rung number for the instruction
		// points to the opcode and mnemonic structure
		OpCode* m_pOpCode;
		// points to the screen object related to this instruction
		CDrawObj* m_pDrawObj;
```

```
};

// special 'list' class for this application (requires afxtempl.h)
typedef CTypedPtrList<CObList, CInstructRung*> CInstructRungList;
class CInstructRung : public CObject
{
	DECLARE_SERIAL( CInstructRung );
	public:
		CInstructRung();
		~CInstructRung();
		CInstructRung* AddParallelRung( int nLineNum, int nX, int pos,
			CDrawRect* pMainLine, CDrawRect* pLeftLine );
		void CompleteParallelRung( int nLineNum, int pos );
		CInstruction* AddInstruction( CDrawObj* pObj );
		BOOL GetOutputInstruction();
		CInstruction* GetInstruction( int nWhich );

		CInstruction* InstructionAtRight( CPLCView* pView, int* pPos );
		CInstruction* InstructionAtLeft( CPLCView* pView, int* pPos, int nX );
		BOOL HasObject( CDrawObj* pObj );
		void Delete( CDrawObj* pObj );
		virtual void Serialize(CArchive& ar);

		CObArray m_arInstructions; // holds all instructions
		BOOL m_bChildRung;	// parallel child rung
		CInstructRung* m_pParentRung;	// parent rung pointer
		CInstructRungList m_arParallelRungs;	// holds all parallel rungs
		int m_nRungNum;		// rung number
		int m_nCurrentInstruct;		// instruction number
		int m_nCurrentX;		// current cursor x position
		int m_nNextObjectX;		// position to place new object at
		CDrawRect* m_pMainRungLine;
		CDrawRect* m_pParallelRungLeftLine;
		CDrawRect* m_pParallelRungRightLine;
};

class CLadder : public CObject
{
	DECLARE_SERIAL( CLadder );
	public:
		CLadder(){ m_nCurrentRungNum = 0; }
		~CLadder();
		CInstructRung* AddRung( CDrawRect* pLine, BOOL bBeforeEnd = FALSE );
		CInstructRung* GetRung( int nRungNum );
		CInstructRung* GetRung( CDrawObj* pObj );
		int GetNextRungNumber(){ return m_nCurrentRungNum++; }
		void DeleteRung( CInstructRung* pRung );

		CInstructRung* GetNearestRungBelow( int nY, CPLCView* pView,
			CInstructRung* pCurrentRung );
		CInstructRungList m_arInstructionRungs;
		virtual void Serialize(CArchive& ar);

	private:
```

```
                CInstructRung* CheckRung( int nRungNum, CInstructRungList* pCheckRung );
                CInstructRung* CheckRung( CDrawCbj* pObj, CInstructRungList* pCheckRung );
                CInstructRung* FindClosestRung( int nY, CInstructRungList* pCheckRung,
                        CPLCView* pView, CInstructRung* pCurrentRung );
                void FixupParents();        // used for serialization
                int m_nCurrentRungNum; // used to generate rung numbers
};
#endif
```

// Instruct.cpp

```
// instruct.cpp : implementation of the instruction generation
#include "stdafx.h"
#include <afxpriv.h>
#include "plcapp.h"
#include "instruct.h"
#include "plcobj.h"
#include "plcView.h"
#include "plcdoc.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

OpCode PLC_OpCodes[] =
{
        // DrawShape drawShape, char szMnemonic[4], int    nOpCode;
        { serial_open,      "AND", 0xA1},
        { serial_open,      "LD",  0xA0},
        { serial_closed,    "ADI", 0xAE},
        { serial_closed,    "NOT", 0xA4},

        { parallel_open,        "OR",   0xA2},
        { parallel_closed,      "ORI",  0xAD},
        { plc_output,           "OUT",  0xAA},
        { reset_switch,         "RSET", 0xB9},
        { set_switch,           "SET",  0xB8},
        { plc_move,             "MOV",          0xBD},
        { plus,                 "ADD",  0xC0},
        { minus,                "SUB",  0xC1},

        { greater_than,         "JG",   0xF1},
        { less_than,            "JL",   0xF2},
        { equal_to,                     "JE",   0xF3},
        { plc_counter,          "CON",  0xF4},
        { plc_timer,            "TIM",  0xF5},
        { plc_start,            "STR",  0xF6},
        { plc_end,              "END",  0xF7},

        { shunt,                "STA",  0xF8},
        { shunt,                "ORS",  0xF9},
        { shunt,                "LDS",  0xFA},
```

```
};

IMPLEMENT_SERIAL( CInstructRung, CObject, 0 );

CInstructRung* CLadder::AddRung( CDrawRect* pMainLine, BOOL bBeforeEnd )
{
	CInstructRung* pRung = new CInstructRung;
	if( bBeforeEnd )
	{
		POSITION pos = m_arInstructionRungs.GetTailPosition();
		m_arInstructionRungs.InsertBefore( pos,  pRung );
	}
	else
		m_arInstructionRungs.AddTail( pRung );
	pRung->m_nRungNum = m_nCurrentRungNum++;
	pRung->m_pMainRungLine = pMainLine;
	return pRung;
}

CInstructRung::CInstructRung()
{
	m_nCurrentInstruct = 0;
	m_nCurrentX = FIRST_COMPONENT_X_VALUE;
	m_bChildRung = FALSE;
	m_nNextObjectX = FIRST_COMPONENT_X_VALUE;
	m_pParentRung = NULL;
	m_pMainRungLine = NULL;
	m_pParallelRungLeftLine = NULL;
	m_pParallelRungRightLine = NULL;
}

CInstruction* CInstructRung::GetInstruction( int nWhich )
{
	if( nWhich < m_arInstructions.GetSize() )
		return (CInstruction*)m_arInstructions[nWhich];
	return NULL;
}

void CInstructRung::Serialize(CArchive& ar)
{
	CObject::Serialize(ar);
	m_arInstructions.Serialize(ar);
	if (ar.IsStoring())
	{
		ar << (WORD)m_bChildRung;
		ar << (WORD)m_nRungNum;
		ar << (WORD)m_nCurrentInstruct;
		ar << (WORD)m_nCurrentX;
		ar << (WORD)m_nNextObjectX;
		ar << (WORD)m_pMainRungLine->m_nLineNumber;
		if( m_pParallelRungLeftLine )
			ar << (DWORD)m_pParallelRungLeftLine->m_nLineNumber;
		else
			ar << (DWORD)-1;
```

```
		if( m_pParallelRungRightLine )
			ar << (DWORD)m_pParallelRungRightLine->m_nLineNumber;
		else
			ar << (DWORD)-1;
		if( m_pParentRung )
			ar << (DWORD)m_pParentRung->m_nRungNum;
		else
			ar << (DWORD)-1;
	}
	else
	{
		WORD wTemp;
		ar >> wTemp; m_bChildRung = wTemp;
		ar >> wTemp; m_nRungNum = wTemp;
		ar >> wTemp; m_nCurrentInstruct = wTemp;
		ar >> wTemp; m_nCurrentX = wTemp;
		ar >> wTemp; m_nNextObjectX = wTemp;

		ar >> wTemp; m_pMainRungLine = (CDrawRect*)wTemp;
		DWORD dwTemp;

		ar >> dwTemp;
		if( dwTemp == -1 )
			m_pParallelRungLeftLine = NULL;
		else
			m_pParallelRungLeftLine = (CDrawRect*)dwTemp;
		ar >> dwTemp;
		if( dwTemp == -1 )
			m_pParallelRungRightLine = NULL;
		else
			m_pParallelRungRightLine = (CDrawRect*)dwTemp;
		ar >> dwTemp;
		if( dwTemp == -1 )
			m_pParentRung = NULL;
		else
			m_pParentRung = (CInstructRung*)m_nRungNum;
	}
	m_arParallelRungs.Serialize(ar);
}

CInstruction* CInstructRung::AddInstruction( CDrawObj* pObj )
{
	CInstruction* pInstruct = new CInstruction( pObj,
		m_nCurrentInstruct );

	m_arInstructions.Add( pInstruct );
	m_nCurrentInstruct++;
	return pInstruct;
}

BOOL CInstructRung::HasObject( CDrawObj* pObj )
{
	CInstruction* pInstruct;
```

```
	int nSize = m_arInstructions.GetSize();
	for( int i = 0; i < nSize; i++ )
	{
		pInstruct = (CInstruction*)m_arInstructions[i];
		if( pInstruct->m_pDrawObj == pObj )
			return TRUE;
	}
	return FALSE;
}

BOOL CInstructRung::GetOutputInstruction()
{
	CInstruction* pInstruct;
	int nSize = m_arInstructions.GetSize();

	for( int i = 0; i < nSize; i++ )
	{
		pInstruct = (CInstruction*)m_arInstructions[i];
		if( pInstruct->GetDrawShape() == plc_output)
			return TRUE;
	}
	return FALSE;
}

CInstruction* CInstructRung::InstructionAtRight( CPLCView* pView, int* pPos )
{
	ASSERT( pPos );
	CInstruction* pInstruct;
	int nSize = m_arInstructions.GetSize();

	for( int i = 0; i < nSize; i++ )
	{
		pInstruct = (CInstruction*)m_arInstructions[i];
		if( pInstruct->m_pDrawObj )
		{
			CRect rect = pInstruct->m_pDrawObj->m_position;
			pView->DocToClient( rect );
			if(	rect.left >= pView->GetDocument()->
				m_cursor.m_nCurrentLineX )
			{
				*pPos = i;
				return pInstruct;
			}
		}
	}
	return NULL;
}

CInstruction* CInstructRung::InstructionAtLeft( CPLCView* pView, int* pPos, int nX )
{
	ASSERT( pPos );
	CInstruction* pInstruct;
	int nSize = m_arInstructions.GetSize();
```

```
	for( int i = nSize-1; i >= 0; i-- )
	{
		pInstruct = (CInstruction*)m_arInstructions[i];
		if( pInstruct->m_pDrawObj )
		{
			CRect rect = pInstruct->m_pDrawObj->m_position;
			pView->DocToClient( rect );
			if(	rect.left <= nX )
			{
				*pPos = i;
				return pInstruct;
			}
		}
	}
	return NULL;
}

CInstructRung* CInstructRung::AddParallelRung( int nLineNum,
	int nX, int pos, CDrawRect* pMainLine,
	CDrawRect* pLeftLine )
{
	CInstructRung* pRung = new CInstructRung;
	m_arParallelRungs.AddTail( pRung );
	pRung->m_nRungNum = nLineNum;
	pRung->m_bChildRung = TRUE;
	pRung->m_nCurrentX = nX;
	pRung->m_nNextObjectX = nX;
	pRung->m_pMainRungLine = pMainLine;
	pRung->m_pParallelRungLeftLine = pLeftLine;
	CInstruction* pInstruct = new CInstruction( TRUE,  nLineNum, pRung );
	m_arInstructions.InsertAt( pos, pInstruct );
	return pRung;
}

void CInstructRung::CompleteParallelRung( int nLineNum, int pos )
{
	CInstruction* pInstruct = new CInstruction( FALSE,  nLineNum, NULL );
	m_arInstructions.InsertAt( pos+1, pInstruct );
}

void CInstructRung::Delete( CDrawObj* pObj )
{
	CInstruction* pInstruct;
	int nSize = m_arInstructions.GetSize();

	for( int i = nSize-1; i >= 0; i-- )
	{
		pInstruct = (CInstruction*)m_arInstructions[i];
		if( pInstruct->m_pDrawObj == pObj)
		{
			m_arInstructions.RemoveAt(i);
			delete pInstruct;
		}
	}
```

```
}

CInstructRung::~CInstructRung()
{
	CInstruction* pInstruct;

	int nSize = m_arInstructions.GetSize();
	for( int i = 0; i < nSize; i++ )
	{
		pInstruct = (CInstruction*)m_arInstructions[i];
		delete pInstruct;
	}
	m_arInstructions.RemoveAll();
	CInstructRung* pRung;
	POSITION pos = m_arParallelRungs.GetHeadPosition();
	while( pos )
	{
		pRung = m_arParallelRungs.GetNext( pos );
		delete pRung;
	}
	m_arParallelRungs.RemoveAll();
}

IMPLEMENT_SERIAL( CInstruction, CObject, 0);

OpCode* CInstruction::GetOpCode( )
{
	int nSize = sizeof(PLC_OpCodes)/sizeof(OpCode);
	if( m_pDrawObj->m_drawShape == serial_open && m_nComponent == 0 )
		return &PLC_OpCodes[1];
	if( m_pDrawObj->m_drawShape == serial_closed && m_nComponent == 0 )
		return &PLC_OpCodes[3];
	for( int i = 0; i < nSize; i++ )
	{
		if( PLC_OpCodes[i].drawShape == m_pDrawObj->m_drawShape )
			return &PLC_OpCodes[i];
	}
	return NULL;
}

CInstruction::CInstruction( CDrawObj* pObj, int nComponentNumber )
{
	m_nComponent = nComponentNumber;
	m_pDrawObj = pObj;
	m_pOpCode = GetOpCode();
	m_Type = Instruction;
	m_pChildRung = NULL;
}

CInstruction::CInstruction() // default constructor
{
	m_Type = Instruction;
	m_pChildRung = NULL;
	m_nComponent = -1;
```

```
}

CInstruction::CInstruction( BOOL bParallelRungBegin, int nRungNumber,
        CInstructRung* pChild )
{
        m_pChildRung = pChild;
        m_pDrawObj = NULL;
        m_pOpCode = NULL;
        m_nComponent = nRungNumber;
        m_Type = bParallelRungBegin ? ParallelBegin : ParallelEnd;
}

void CInstruction::Serialize(CArchive& ar)
{
        CObject::Serialize(ar);
        if (ar.IsStoring())
        {
                ar << (WORD)m_nRung;
                ar << (WORD)m_nComponent;
                ar << (WORD)m_Type;
                if( m_pDrawObj )
                        ar << (WORD)m_pDrawObj->m_nObjectNumber;
                else
                if( m_Type == ParallelBegin )
                        ar << (WORD)m_pChildRung->m_nRungNum;
        }
        else
        {
                WORD wTemp;
                ar >> wTemp; m_nRung= wTemp;
                ar >> wTemp; m_nComponent = wTemp;
                ar >> wTemp; m_Type = (InstructionType)wTemp;

                if( m_Type == Instruction )
                {
                        ar >> wTemp;
                        // use this value later to get a pointer to m_pDrawObj
                        m_pDrawObj = (CDrawObj*)wTemp;
                        m_pChildRung = NULL;
                }
                else
                if( m_Type == ParallelBegin )
                {
                        // get the rung number
                        ar >> wTemp;
                        // use this value to get a ptr to rung
                        m_pChildRung = (CInstructRung*)wTemp;
                        m_pDrawObj = NULL;
                }
                else
                if( m_Type == ParallelEnd )
                {
                        m_pChildRung = NULL;
                        m_pDrawObj = NULL;
```

```
		}
	}
}

IMPLEMENT_SERIAL( CLadder, CObject, 0);
CLadder::~CLadder()
{
	CInstructRung* pRung;
	POSITION pos = m_arInstructionRungs.GetHeadPosition();
	while( pos )
	{
		pRung = m_arInstructionRungs.GetNext(pos);
		delete pRung;
	}
	m_arInstructionRungs.RemoveAll();
}

void CLadder::DeleteRung( CInstructRung* pRung )
{
	POSITION pos = m_arInstructionRungs.GetHeadPosition();
	while( pos )
	{
		POSITION oldPos = pos;
		CInstructRung* pRungFound = m_arInstructionRungs.GetNext(pos);
		if( pRung == pRungFound )
		{
			m_arInstructionRungs.RemoveAt( oldPos );
			delete pRung;
			return;
		}

		// iterate for the parallel rung
		if( pRungFound->m_arParallelRungs.GetCount() )
		{
			POSITION parallelPos =
				pRungFound->m_arParallelRungs.GetHeadPosition();
			while( parallelPos )
			{
				POSITION oldParallelPos = parallelPos;
				CInstructRung* pParallelRungFound = pRungFound->
					m_arParallelRungs.GetNext(parallelPos);
				if( pRung == pParallelRungFound )
				{
					pRungFound->m_arParallelRungs.RemoveAt
						( oldParallelPos );
					delete pRung;
					return;
				}
			}
		}
	}
	ASSERT(FALSE);
}
```

```
CInstructRung* CLadder::GetNearestRungBelow( int nY, CPLCView* pView,
        CInstructRung* pCurrentRung )
{
        return FindClosestRung( nY, &m_arInstructionRungs,
                pView, pCurrentRung );
}

CInstructRung* CLadder::FindClosestRung( int nY,
        CInstructRungList* pCheckRung,
        CPLCView* pView, CInstructRung* pCurrentRung )
{
        CInstructRung* pRung;
        POSITION posRung = pCheckRung->GetHeadPosition();
        while( posRung )
        {
                pRung = pCheckRung->GetNext(posRung);
                CRect rect = pRung->m_pMainRungLine->m_position;
                pView->DocToClient( rect );

                // if both rungs being compared are children of the
                // same parent, ignore them
                if( pCurrentRung->m_bChildRung && pRung->m_bChildRung &&
                        pCurrentRung->m_pParentRung == pRung->m_pParentRung )
                        goto skip;
                else
                if( abs(rect.top - nY) < MINIMUM_DIST_BET_RUNG )
                        return pRung;

                skip :
                // do a recursive call for the parallel rung
                if( pRung->m_arParallelRungs.GetCount() )
                {
                        CInstructRung* pParallelRung = FindClosestRung( nY,
                                &pRung->m_arParallelRungs, pView, pCurrentRung );

                        if( pParallelRung )
                                return pParallelRung;
                }
        }

        return NULL;
}

CInstructRung* CLadder::GetRung( CDrawObj* pObj )
{
        return CheckRung( pObj, &m_arInstructionRungs );
}

CInstructRung* CLadder::CheckRung( CDrawObj* pObj, CInstructRungList* pCheckRung )
{
        CInstructRung* pRung;
        POSITION posRung = pCheckRung->GetHeadPosition();
        while( posRung )
        {
```

```
		pRung = pCheckRung->GetNext(posRung);
		if( pRung->HasObject( pObj ) )
			return pRung;

		// do a recursive call for the parallel rung
		if( pRung->m_arParallelRungs.GetCount() )
		{
			CInstructRung* pParallelRung = CheckRung( pObj,
				&pRung->m_arParallelRungs );
			if( pParallelRung )
				return pParallelRung;
		}
	}
	return NULL;
}

CInstructRung* CLadder::GetRung( int nRungNum )
{
	return CheckRung( nRungNum, &m_arInstructionRungs );
}

CInstructRung* CLadder::CheckRung( int nRungNum, CInstructRungList* pCheckRung )
{
	CInstructRung* pRung;
	POSITION posRung = pCheckRung->GetHeadPosition();
	while( posRung )
	{
		pRung = pCheckRung->GetNext(posRung);
		if( pRung->m_nRungNum == nRungNum )
			return pRung;

		// do a recursive call for the parallel rung
		if( pRung->m_arParallelRungs.GetCount() )
		{
			CInstructRung* pParallelRung = CheckRung( nRungNum,
				&pRung->m_arParallelRungs );
			if( pParallelRung )
				return pParallelRung;
		}
	}
	return NULL;
}

void CLadder::Serialize(CArchive& ar)
{
	CObject::Serialize(ar);

	if (ar.IsStoring())
	{
		m_arInstructionRungs.Serialize(ar);
		ar << (WORD)m_nCurrentRungNum;
	}
	else
	{
```

```
                m_arInstructionRungs.Serialize(ar);
                WORD wTemp;
                ar >> wTemp; m_nCurrentRungNum = wTemp;

                // fixup the parent rung pointers
                FixupParents();
        }
}

void CLadder::FixupParents()
{
        CInstructRung* pRung, *pRung2;

        POSITION posRung = m_arInstructionRungs.GetHeadPosition();
        while( posRung )
        {
                pRung = m_arInstructionRungs.GetNext(posRung);

                // only one level deep rungs
                if( pRung->m_arParallelRungs.GetCount() )
                {
                        POSITION posParallelRung = pRung->m_arParallelRungs.
                                GetHeadPosition();

                        while( posParallelRung )
                        {
                                pRung2 = pRung->m_arParallelRungs.GetNext( posParallelRung );
                                pRung2->m_pParentRung = pRung;
                        }
                }
        }
}
```

// instrvw.h : interface of the CInstructView class

```
#ifndef _VW_H
#define _VW_H
#define HINT_APPEND_STR 10000
#define HINT_INIT 10001
#define HINT_CUT_ALL 10002
#include "plcview.h"

// CInstructView
// CStringOb -- CObject container for a CString so it can be passed as an update hint

class CStringOb : public CObject
{
        DECLARE_DYNAMIC(CStringOb)
public:
        CString m_str;
        CStringOb() {}
        CStringOb( const char* szStr ): m_str( szStr ) {}
};
```

```
// CInstructDoc
class CInstructDoc : public CDocument
{
	DECLARE_DYNCREATE(CInstructDoc)

public:
	virtual ~CInstructDoc();
	void AppendToViews( const char* szText );
	void AppendToViews( CStringOb& strObAppend );
	void CutAll();
	virtual BOOL InitializeView( CView* pView );
	virtual void Serialize(CArchive& ar);
	virtual void SetViewPtr( CPLCView* pView  );

#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
	CInstructDoc();
	virtual BOOL OnNewDocument();
	void OnCloseDocument();
	CPLCView* m_pCallingView;

	// Generated message map functions
	DECLARE_MESSAGE_MAP()
};


class CInstructView : public CEditView
{
	DECLARE_DYNCREATE(CInstructView)

protected:
	BOOL m_bInitialized;
public:
	CInstructDoc* GetDocument();

	virtual ~CInstructView();
	virtual void OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint );
	void AppendText( const char* szText );
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
	CInstructView();

	// Printing support
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
```

```
	virtual void OnEndPrinting(CDC* pDC, CPrinInfo* pInfo);

	// Generated message map functions
	afx_msg void OnEditSetFocus();
	afx_msg long OnInitialize( WPARAM wParam, LPARAM lParam );
	DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG
inline CInstructDoc* CInstructView::GetDocument()
  { return (CInstructDoc*)m_pDocument; }
#endif

class CInstructFrame : public CMDIChildWnd
{
	DECLARE_DYNCREATE(CInstructFrame )
public:
	CInstructFrame();

};
#endif
```

// instrvw.cpp : implementation of the CInstructView class

```
#include "stdafx.h"
#include "plcapp.h"
#include "plcdoc.h"
#include "instrvw.h"
#include "plcview.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

#define WM_USER_INITIALIZE (WM_USER+1000)

IMPLEMENT_DYNCREATE(CInstructView, CEditView)
IMPLEMENT_DYNAMIC(CStringOb, CObject)
BEGIN_MESSAGE_MAP(CInstructView, CEditView)
	ON_EN_SETFOCUS(AFX_IDW_PANE_FIRST, OnEditSetFocus)
	ON_MESSAGE(WM_USER_INITIALIZE, OnInitialize)
	ON_COMMAND(ID_FILE_PRINT, CEditView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CEditView::OnFilePrintPreview)
END_MESSAGE_MAP()
CInstructView::CInstructView() : m_bInitialized( FALSE )
{
}

CInstructView::~CInstructView()
{
}

void CInstructView::OnEditSetFocus()
```

```
{
	if( !m_bInitialized )
	{
		// complete set focus then initialize
		PostMessage( WM_USER_INITIALIZE );
	}
}

long CInstructView::OnInitialize( WPARAM wParam, LPARAM lParam )
{
	if( !m_bInitialized )
	{
		// if not the first CInstructView for this doc,
		// copy contents from first view
		CInstructDoc* pDoc = GetDocument();
		POSITION posView = pDoc->GetFirstViewPosition();
		while( posView )
		{
			CView* pNextView = pDoc->GetNextView( posView );
			ASSERT( pNextView );
			if( pNextView != this && pNextView->IsKindOf(
				RUNTIME_CLASS( CInstructView )))
			{
				CString strTemp;
				((CInstructView*)pNextView)->GetEditCtrl().
					GetWindowText( strTemp );
				GetEditCtrl().SetWindowText( strTemp );
				m_bInitialized = TRUE;

			  return 0L;
			}
		}
		m_bInitialized = TRUE;
	}
  return 0L;
}

/* virtual */ void CInstructView::OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint )
{
	switch( lHint )
	{
		case HINT_INIT:
			break;
		case HINT_APPEND_STR:
			AppendText( ((CStringOb*)pHint)->m_str );
			return;
		case HINT_CUT_ALL :
			OnEditSelectAll();
			OnEditCut();
			break;
		 default:
			break;	// unexpected hint
	}
	CEditView::OnUpdate( pSender, lHint, pHint );
```

```
}

void CInstructView::AppendText( const char* szText )
{
	DWORD curSel = GetEditCtrl().GetSel();
	int textLenght = GetEditCtrl().GetWindowTextLength();
	GetEditCtrl().SetSel( textLenght, textLenght );
	GetEditCtrl().ReplaceSel( szText );
	GetEditCtrl().SetSel( curSel, TRUE );
}

/* virtual */ void CInstructView::OnDraw(CDC* pDC)
{
	CInstructDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	CEditView::OnDraw( pDC );
}

/* virtual */ BOOL CInstructView::OnPreparePrinting(CPrintInfo* pInfo)
{
	return CEditView::OnPreparePrinting(pInfo);
}

/* virtual */ void CInstructView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
	CEditView::OnBeginPrinting(pDC, pInfo);
}

/* virtual */ void CInstructView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
	CEditView::OnEndPrinting(pDC, pInfo);
}

#ifdef _DEBUG
/* virtual */ void CInstructView::AssertValid() const
{
	CEditView::AssertValid();
}

/* virtual */ void CInstructView::Dump(CDumpContext& dc) const
{
	CEditView::Dump(dc);
}

CInstructDoc* CInstructView::GetDocument()
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CInstructDoc)));
	return (CInstructDoc*)m_pDocument;
}
#endif //_DEBUG

// CInstructFrame
IMPLEMENT_DYNCREATE(CInstructFrame, CMDIChildWnd)
```

```
CInstructFrame::CInstructFrame()
{
}

// CInstructDoc
void CInstructDoc::AppendToViews( const char* szText )
{
	CStringOb strObAppend( szText );
	AppendToViews( strObAppend );
}

void CInstructDoc::AppendToViews( CStringOb& strObAppend )
{
	UpdateAllViews( NULL, HINT_APPEND_STR, &strObAppend );
}

void CInstructDoc::CutAll()
{
	UpdateAllViews( NULL, HINT_CUT_ALL, NULL );
}

IMPLEMENT_DYNCREATE(CInstructDoc, CDocument)
BEGIN_MESSAGE_MAP(CInstructDoc, CDocument)
END_MESSAGE_MAP()
CInstructDoc::CInstructDoc()
{
}

CInstructDoc::~CInstructDoc()
{
}

/* virtual */BOOL CInstructDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;
	return TRUE;
}

void CInstructDoc::SetViewPtr( CPLCView* pView )
{
	m_pCallingView = pView;
}

void CInstructDoc::OnCloseDocument()
{
	m_pCallingView->ResetDocPtr();
	CDocument::OnCloseDocument();
}

/* virtual */BOOL CInstructDoc::InitializeView( CView* pView )
{
	// derived classes which implement this method should not
	// call this base function
```

```
	return FALSE;
}

/* virtual */void CInstructDoc::Serialize(CArchive& ar)
{
	CView* activeView = (CView*)m_viewList.GetHead();
	CFrameWnd* activeFrame = ((CMDIFrameWnd*)AfxGetApp()->
		m_pMainWnd)->GetActiveFrame();
	if( activeFrame )
	{
		if( activeFrame->GetActiveDocument() == this )
			activeView = activeFrame->GetActiveView();
	}
	if( activeView && activeView->IsKindOf( RUNTIME_CLASS( CEditView )))
	((CEditView*)activeView)->SerializeRaw(ar);
}

#ifdef _DEBUG
/* virtual */ void CInstructDoc::AssertValid() const
{
	CDocument::AssertValid();
}

/* virtual */ void CInstructDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG
```

// Resource.h

```
//{{NO_DEPENDENCIES}}
#define IDR_MAINFRAME		2
#define IDS_STRING_COMPLETE_SHUNT	4
#define IDS_STRING_OBJECT_EXISTS	5
#define IDR_INSTR_TYPE		7
#define IDR_PLCTYPE		8
#define IDR_PLCTYPE_CNTR_IP		9
#define IDD_ABOUTBOX		100
#define IDP_BUSY		100
#define IDP_OLE_INIT_FAILED		101
#define IDD_PROP_RECT		101
#define IDP_FAILED_TO_CREATE		102
#define IDC_WEIGHT		102
#define IDC_PENCIL		102
#define IDC_NOFILL		103
#define IDB_BITMAP_SERIAL_OPEN_SW	103
#define IDC_EDIT_NAME		104
#define IDC_STATIC_VALUE		105
#define IDB_BITMAP_SERIAL_CLOSED_SW	106
#define IDC_EDIT_NAME2		106
#define IDB_BITMAP_OUTPUT		107
#define IDC_STATIC_VALUE2		107
```

```
#define IDB_BITMAP_RESET            108
#define IDB_BITMAP_SET              109
#define IDB_BITMAP_MOVE             110
#define IDB_BITMAP_GREATER_THAN     111
#define IDB_BITMAP_LESS_THAN        112
#define IDB_BITMAP_EQUAL_TO         113
#define IDB_BITMAP_ADDITION         114
#define IDB_BITMAP_SUBTRACTION      115
#define IDB_BITMAP_TIMER            116
#define IDB_BITMAP_COUNTER          117
#define IDD_DIALOG_OBJECT_NAME      117
#define IDB_BITMAP_END              118
#define IDD_DIALOG_OBJECT_NAMES     118
#define IDB_BITMAP_START            119
#define IDB_BITMAP_SERIAL_OPEN_SW1  120
#define ID_CANCEL_EDIT              32768
#define ID_VIEW_GRID                32769
#define ID_DRAW_SELECT              32770
#define ID_DRAW_LINE                32771
#define ID_DRAW_RECT                32772
#define ID_DRAW_ROUNDRECT           32773
#define ID_DRAW_ELLIPSE             32774
#define ID_OBJECT_LINECOLOR         32775
#define ID_OBJECT_FILLCOLOR         32776
#define ID_OBJECT_MOVETOFRONT       32777
#define ID_OBJECT_MOVETOBACK        32778
#define ID_OBJECT_MOVEFORWARD       32779
#define ID_OBJECT_MOVEBACK          32780
#define ID_VIEW_PAPERCOLOR          32781
#define ID_DRAW_POLYGON             32782
#define ID_VIEW_SHOWOBJECTS         32783
#define ID_EDIT_PROPERTIES          32784
#define ID_DRAW_SERIAL_OPEN_SWITCH  32785
#define IDS_ILLEGAL_OUTPUT_POSITION 32786
#define IDS_STRING_BAD_POINT        32787
#define IDS_STRING_BACKWARD_POINT   32788
#define IDS_INCOMPLETE_RUNG         32789
#define ID_DRAW_SERIAL_CLOSED_SWITCH 32790
#define IDS_STRING_NO_COMPONENT     32791
#define IDS_BAD_CURSOR_POS          32792
#define ID_DRAW_RESET_SWITCH        32793
#define ID_DRAW_OUTPUT              32794
#define ID_DRAW_SET_SWITCH          32795
#define ID_STR_INSERT_RUNG          32795
#define ID_DRAW_MOVE                32796
#define ID_DRAW_GREATERTHAN         32797
#define ID_DRAW_LESSTHAN            32798
#define ID_DRAW_EQUALTO             32799
#define ID_DRAW_ADDITION            32800
#define ID_DRAW_SUBTRACTION         32801
#define ID_DRAW_START               32802
#define ID_DRAW_STOP                32803
#define ID_DRAW_COUNTER             32804
#define ID_DRAW_TIMER               32805
```

```
#define ID_PLC_GENERATECODE          32806
#define IDS_ILLEGAL_CURSOR_POSITION    32806
#define ID_DRAW_SHUNT              32807
#define ID_DRAW_UPWARD_SHUNT         32808
#define ID_EDIT_DELETERUNG           32809
#define IDS_ILLEGAL_SWITCH_POSITION    61446
#define IDS_ILLEGAL_SWITCH_POSITION_FORENDRUNG 61447

// Next default values for new objects
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE       119
#define _APS_NEXT_COMMAND_VALUE        32810
#define _APS_NEXT_CONTROL_VALUE        107
#define _APS_NEXT_SYMED_VALUE          101
#endif
#endif
```

// PLCApp.Rc

```
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
#include "afxres.h"
#undef APSTUDIO_READONLY_SYMBOLS
#ifdef APSTUDIO_INVOKED
// TEXTINCLUDE

1 TEXTINCLUDE DISCARDABLE
BEGIN
   "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
   "#include ""afxres.h""\r\n"
   "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
   "#include ""res\\drawcli.rc2""  // non-App Studio edited resources\r\n"
   "\r\n"
   "#include ""afxres.rc""  // Standard components\r\n"
   "#include ""afxprint.rc""  // printing/print preview resources\r\n"
   "#include ""afxolecl.rc""  // OLE container resources\r\n"
   "\0"
END

#endif   // APSTUDIO_INVOKED
// Icon
IDR_MAINFRAME        ICON   DISCARDABLE    "RES\\DRAWCLI.ICO"
IDR_DRAWCLTYPE       ICON   DISCARDABLE    "RES\\DRAWDOC.ICO"
```

```
// Bitmap
IDR_MAINFRAME          BITMAP  MOVEABLE PURE   "RES\\TOOLBAR.BMP"
IDB_BITMAP_SERIAL_OPEN_SW BITMAP  DISCARDABLE    "RES\\BITMAP1.BMP"
IDB_BITMAP_SERIAL_CLOSED_SW BITMAP  DISCARDABLE    "RES\\BITMAP_S.BMP"
IDB_BITMAP_OUTPUT      BITMAP  DISCARDABLE    "RES\\BITMAP_O.BMP"
IDB_BITMAP_RESET       BITMAP  DISCARDABLE    "RES\\BITMAP_R.BMP"
IDB_BITMAP_SET         BITMAP  DISCARDABLE    "RES\\BMP00002.BMP"
IDB_BITMAP_MOVE        BITMAP  DISCARDABLE    "RES\\BMP00003.BMP"
IDB_BITMAP_GREATER_THAN BITMAP  DISCARDABLE    "RES\\BITMAP_M.BMP"
IDB_BITMAP_LESS_THAN   BITMAP  DISCARDABLE    "RES\\BITMAP_G.BMP"
IDB_BITMAP_EQUAL_TO    BITMAP  DISCARDABLE    "RES\\BITMAP_L.BMP"
IDB_BITMAP_ADDITION    BITMAP  DISCARDABLE    "RES\\BITMAP_A.BMP"
IDB_BITMAP_TIMER       BITMAP  DISCARDABLE    "RES\\BITMAP_C.BMP"
IDB_BITMAP_COUNTER     BITMAP  DISCARDABLE    "RES\\BITMAP_T.BMP"
IDB_BITMAP_END         BITMAP  DISCARDABLE    "RES\\BMP00005.BMP"
IDB_BITMAP_START       BITMAP  DISCARDABLE    "RES\\BMP00006.BMP"
IDB_BITMAP_SUBTRACTION BITMAP  DISCARDABLE    "RES\\BMP00004.BMP"

// Menu
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New\tCtrl+N",          ID_FILE_NEW
    MENUITEM "&Open...\tCtrl+O",      ID_FILE_OPEN
    MENUITEM SEPARATOR
    MENUITEM "P&rint Setup...",       ID_FILE_PRINT_SETUP
    MENUITEM SEPARATOR
    MENUITEM "Recent File",           ID_FILE_MRU_FILE1, GRAYED
    MENUITEM SEPARATOR
    MENUITEM "E&xit",                 ID_APP_EXIT
  END
  POPUP "&View"
  BEGIN
    MENUITEM "&Toolbar",              ID_VIEW_TOOLBAR
    MENUITEM "&Status Bar",           ID_VIEW_STATUS_BAR
  END
  POPUP "&Help"
  BEGIN
    MENUITEM "&About PLC Circuit Builder...", ID_APP_ABOUT
  END
END

IDR_DRAWCLTYPE MENU PRELOAD DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New\tCtrl+N",          ID_FILE_NEW
    MENUITEM "&Open...\tCtrl+O",      ID_FILE_OPEN
    MENUITEM "&Close",                ID_FILE_CLOSE
    MENUITEM "&Save\tCtrl+S",         ID_FILE_SAVE
    MENUITEM "Save &As...",           ID_FILE_SAVE_AS
    MENUITEM SEPARATOR
    MENUITEM "&Print...\tCtrl+P",     ID_FILE_PRINT
```

```
    MENUITEM "Print Pre&view",          ID_FILE_PRINT_PREVIEW
    MENUITEM "P&rint Setup...",         ID_FILE_PRINT_SETUP
    MENUITEM SEPARATOR
    MENUITEM "Recent File",             ID_FILE_MRU_FILE1, GRAYED
    MENUITEM SEPARATOR
    MENUITEM "E&xit",                   ID_APP_EXIT
  END
  POPUP "&Edit"
  BEGIN
    MENUITEM "&Undo\tCtrl+Z",           ID_EDIT_UNDO
    MENUITEM SEPARATOR
    MENUITEM "Cu&t\tCtrl+X",            ID_EDIT_CUT
    MENUITEM "&Copy\tCtrl+C",           ID_EDIT_COPY
    MENUITEM "&Paste\tCtrl+V",          ID_EDIT_PASTE
    MENUITEM "&Delete\tDel",            ID_EDIT_CLEAR
    MENUITEM "Select &All\tCtrl+A",     ID_EDIT_SELECT_ALL
    MENUITEM "Proper&ties...\tAlt+Enter", ID_EDIT_PROPERTIES
    MENUITEM SEPARATOR
    MENUITEM "Insert &New Object...",   ID_OLE_INSERT_NEW
    MENUITEM "Lin&ks...",               ID_OLE_EDIT_LINKS
    MENUITEM "<<OLE VERBS GO HERE>>",   ID_OLE_VERB_FIRST
  END
  POPUP "&View"
  BEGIN
    MENUITEM "&Grid Lines\tG",          ID_VIEW_GRID
    MENUITEM "&Paper Color...",         ID_VIEW_PAPERCOLOR
    MENUITEM "Show &Objects",           ID_VIEW_SHOWOBJECTS
    MENUITEM SEPARATOR
    MENUITEM "&Toolbar",                ID_VIEW_TOOLBAR
    MENUITEM "&Status Bar",             ID_VIEW_STATUS_BAR
  END
  POPUP "&Draw"
  BEGIN
    MENUITEM "&Select",                 ID_DRAW_SELECT
    MENUITEM "&New Rung",               ID_DRAW_LINE
    MENUITEM "Sh&unt",                  ID_DRAW_SHUNT
    MENUITEM "&Complete Parallel Rung", ID_DRAW_UPWARD_SHUNT
    POPUP "S&witch"
    BEGIN
      MENUITEM "&Open Switch",          ID_DRAW_SERIAL_OPEN_SWITCH

      MENUITEM "Close&d Switch",        ID_DRAW_SERIAL_CLOSED_SWITCH

      MENUITEM "&Reset Switch",         ID_DRAW_RESET_SWITCH
      MENUITEM "S&et Switch",           ID_DRAW_SET_SWITCH
      MENUITEM "Outpu&t",               ID_DRAW_OUTPUT
    END
    POPUP "&Instruction"
    BEGIN
      MENUITEM "Move",                  ID_DRAW_MOVE
      MENUITEM "Greater Than",          ID_DRAW_GREATERTHAN
      MENUITEM "Less Than",             ID_DRAW_LESSTHAN
      MENUITEM "Equal To",              ID_DRAW_EQUALTO
      MENUITEM "Addition",              ID_DRAW_ADDITION
```

```
        MENUITEM "Subtraction",             ID_DRAW_SUBTRACTION
      END
      POPUP "&Advanced Instruction"
      BEGIN
        MENUITEM "Counter",                 ID_DRAW_COUNTER
        MENUITEM "Timer",                   ID_DRAW_TIMER
      END
      MENUITEM "S&tart",                    ID_DRAW_START
      MENUITEM "Sto&p",                     ID_DRAW_STOP
    END
    POPUP "&Object"
    BEGIN
      MENUITEM "&Line Color...",            ID_OBJECT_LINECOLOR
      MENUITEM "&Fill Color...",            ID_OBJECT_FILLCOLOR
      MENUITEM SEPARATOR
      MENUITEM "Move To Front\tCtrl+Plus",   ID_OBJECT_MOVETOFRONT
      MENUITEM "Move To Back\tCtrl+Minus",   ID_OBJECT_MOVETOBACK
      MENUITEM "Move Forward\tPlus",        ID_OBJECT_MOVEFORWARD
      MENUITEM "Move Back\tMinus",          ID_OBJECT_MOVEBACK
    END
    POPUP "&PLC"
    BEGIN
      MENUITEM "&Generate Code",            ID_PLC_GENERATECODE
    END
    POPUP "&Window"
    BEGIN
      MENUITEM "&New Window",               ID_WINDOW_NEW
      MENUITEM "&Cascade",                  ID_WINDOW_CASCADE
      MENUITEM "&Tile",                     ID_WINDOW_TILE_HORZ
      MENUITEM "&Arrange Icons",            ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
    BEGIN
      MENUITEM "&About PLC Circuit Builder...", ID_APP_ABOUT
    END
  END

IDR_DRAWCLTYPE_CNTR_IP MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
      MENUITEM "&New\tCtrl+N",              ID_FILE_NEW
      MENUITEM "&Open...\tCtrl+O",          ID_FILE_OPEN
      MENUITEM "&Close",                    ID_FILE_CLOSE
      MENUITEM "&Save\tCtrl+S",             ID_FILE_SAVE
      MENUITEM "Save &As...",               ID_FILE_SAVE_AS
      MENUITEM SEPARATOR
      MENUITEM "&Print...\tCtrl+P",         ID_FILE_PRINT
      MENUITEM "Print Pre&view",            ID_FILE_PRINT_PREVIEW
      MENUITEM "P&rint Setup...",           ID_FILE_PRINT_SETUP
      MENUITEM SEPARATOR
      MENUITEM "Recent File",               ID_FILE_MRU_FILE1, GRAYED
      MENUITEM SEPARATOR
      MENUITEM "E&xit",                     ID_APP_EXIT
```

```
    END
    MENUITEM SEPARATOR
    MENUITEM SEPARATOR
    POPUP "&Window"
    BEGIN
      MENUITEM "&New Window",            ID_WINDOW_NEW
      MENUITEM "&Cascade",               ID_WINDOW_CASCADE
      MENUITEM "&Tile",                  ID_WINDOW_TILE_HORZ
      MENUITEM "&Arrange Icons",         ID_WINDOW_ARRANGE
    END
  END

  IDR_INSTR_TYPE MENU PRELOAD DISCARDABLE
  BEGIN
    POPUP "&File"
    BEGIN
      MENUITEM "&New\tCtrl+N",           ID_FILE_NEW
      MENUITEM "&Open...\tCtrl+O",       ID_FILE_OPEN
      MENUITEM "&Close",                 ID_FILE_CLOSE
      MENUITEM "&Save\tCtrl+S",          ID_FILE_SAVE
      MENUITEM "Save &As...",            ID_FILE_SAVE_AS
      MENUITEM SEPARATOR
      MENUITEM "&Print...\tCtrl+P",      ID_FILE_PRINT
      MENUITEM "Print Pre&view",         ID_FILE_PRINT_PREVIEW
      MENUITEM "P&rint Setup...",        ID_FILE_PRINT_SETUP
      MENUITEM SEPARATOR
      MENUITEM "Recent File",            ID_FILE_MRU_FILE1, GRAYED
      MENUITEM SEPARATOR
      MENUITEM "E&xit",                  ID_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
      MENUITEM "&Undo\tCtrl+Z",          ID_EDIT_UNDO
      MENUITEM SEPARATOR
      MENUITEM "Cu&t\tCtrl+X",           ID_EDIT_CUT
      MENUITEM "&Copy\tCtrl+C",          ID_EDIT_COPY
      MENUITEM "&Paste\tCtrl+V",         ID_EDIT_PASTE
      MENUITEM "&Delete\tDel",           ID_EDIT_CLEAR
      MENUITEM "Select &All\tCtrl+A",    ID_EDIT_SELECT_ALL
      MENUITEM "Proper&ties...\tAlt+Enter", ID_EDIT_PROPERTIES
      MENUITEM SEPARATOR
    END
    POPUP "&View"
    BEGIN
      MENUITEM "&Toolbar",               ID_VIEW_TOOLBAR
      MENUITEM "&Status Bar",            ID_VIEW_STATUS_BAR
    END
    POPUP "&Window"
    BEGIN
      MENUITEM "&Cascade",               ID_WINDOW_CASCADE
      MENUITEM "&Tile",                  ID_WINDOW_TILE_HORZ
      MENUITEM "&Arrange Icons",         ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
```

```
  BEGIN
    MENUITEM "&About PLC Circuit Builder...", ID_APP_ABOUT
  END
END

// Accelerator
IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
  "+",        ID_OBJECT_MOVETOFRONT,  ASCII,  NOINVERT
  "-",        ID_OBJECT_MOVEBACK,     ASCII,  NOINVERT
  "=",        ID_OBJECT_MOVEFORWARD,  ASCII,  NOINVERT
  "_",        ID_OBJECT_MOVETOBACK,   ASCII,  NOINVERT
  "A",        ID_EDIT_SELECT_ALL,     VIRTKEY, CONTROL, NOINVERT
  "C",        ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
  "E",        ID_DRAW_ELLIPSE,        ASCII,  NOINVERT
  "e",        ID_DRAW_ELLIPSE,        ASCII,  NOINVERT
  "G",        ID_VIEW_GRID,           ASCII,  NOINVERT
  "g",        ID_VIEW_GRID,           ASCII,  NOINVERT
  "L",        ID_DRAW_LINE,           ASCII,  NOINVERT
  "l",        ID_DRAW_LINE,           ASCII,  NOINVERT
  "N",        ID_DRAW_ROUNDRECT,      ASCII,  NOINVERT
  "N",        ID_FILE_NEW,            VIRTKEY, CONTROL, NOINVERT
  "n",        ID_DRAW_ROUNDRECT,      ASCII,  NOINVERT
  "O",        ID_FILE_OPEN,           VIRTKEY, CONTROL, NOINVERT
  "P",        ID_FILE_PRINT,          VIRTKEY, CONTROL, NOINVERT
  "R",        ID_DRAW_RECT,           ASCII,  NOINVERT
  "r",        ID_DRAW_RECT,           ASCII,  NOINVERT
  "S",        ID_DRAW_SELECT,         ASCII,  NOINVERT
  "S",        ID_FILE_SAVE,           VIRTKEY, CONTROL, NOINVERT
  "s",        ID_DRAW_SELECT,         ASCII,  NOINVERT
  "V",        ID_EDIT_PASTE,          VIRTKEY, CONTROL, NOINVERT
  VK_ADD,     ID_OBJECT_MOVEFORWARD,  VIRTKEY, NOINVERT
  VK_ADD,     ID_OBJECT_MOVETOFRONT,  VIRTKEY, CONTROL, NOINVERT
  VK_BACK,    ID_EDIT_UNDO,           VIRTKEY, ALT, NOINVERT
  VK_DELETE,  ID_EDIT_CLEAR,          VIRTKEY, NOINVERT
  VK_DELETE,  ID_EDIT_CUT,            VIRTKEY, SHIFT, NOINVERT
  VK_ESCAPE,  ID_CANCEL_EDIT,         VIRTKEY, NOINVERT
  VK_F6,      ID_NEXT_PANE,           VIRTKEY, NOINVERT
  VK_F6,      ID_PREV_PANE,           VIRTKEY, SHIFT, NOINVERT
  VK_INSERT,  ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
  VK_INSERT,  ID_EDIT_PASTE,          VIRTKEY, SHIFT, NOINVERT
  VK_RETURN,  ID_EDIT_PROPERTIES,     VIRTKEY, ALT, NOINVERT
  VK_SUBTRACT, ID_OBJECT_MOVEBACK,    VIRTKEY, NOINVERT
  VK_SUBTRACT, ID_OBJECT_MOVETOBACK,  VIRTKEY, CONTROL, NOINVERT
  "X",        ID_EDIT_CUT,            VIRTKEY, CONTROL, NOINVERT
  "Z",        ID_EDIT_UNDO,           VIRTKEY, CONTROL, NOINVERT
END

IDR_DRAWCLTYPE_CNTR_IP ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
  "N",        ID_FILE_NEW,            VIRTKEY, CONTROL
  "O",        ID_FILE_OPEN,           VIRTKEY, CONTROL
  "S",        ID_FILE_SAVE,           VIRTKEY, CONTROL
  "P",        ID_FILE_PRINT,          VIRTKEY, CONTROL
```

```
  VK_F6,        ID_NEXT_PANE,        VIRTKEY
  VK_F6,        ID_PREV_PANE,        VIRTKEY, SHIFT
  VK_ESCAPE,    ID_CANCEL_EDIT,      VIRTKEY, NOINVERT
END

// Dialog
IDD_ABOUTBOX DIALOG DISCARDABLE  34, 22, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About PLC Circuit Builder"
FONT 8, "MS Sans Serif"
BEGIN
  ICON          IDR_MAINFRAME,IDC_STATIC,11,17,18,20
  LTEXT         "PLC Circuit Builder Version 1.0",IDC_STATIC,40,10,119,8
  LTEXT         "Copyright \251 Archana More 1996",IDC_STATIC,40,25,119,
            8
  DEFPUSHBUTTON   "OK",IDOK,176,6,32,14,WS_GROUP
END

IDD_PROP_RECT DIALOG DISCARDABLE  20, 20, 185, 92
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Shape Properties"
FONT 8, "MS Sans Serif"
BEGIN
  GROUPBOX      "Outline",IDC_STATIC,6,6,101,33
  LTEXT         "Weight:",IDC_STATIC,11,20,27,8
  EDITTEXT      IDC_WEIGHT,41,18,21,13,ES_AUTOHSCROLL
  GROUPBOX      "Fill",IDC_STATIC,6,44,100,40
  CONTROL       "None",IDC_NOFILL,"Button",BS_AUTOCHECKBOX | WS_TABSTOP,
            11,56,35,10
  DEFPUSHBUTTON   "OK",IDOK,129,6,50,14
  PUSHBUTTON      "Cancel",IDCANCEL,129,23,50,14
END

IDD_DIALOG_OBJECT_NAME DIALOG DISCARDABLE  0, 0, 133, 57
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Object Property"
FONT 8, "MS Sans Serif"
BEGIN
  LTEXT         "Name Of Object:",IDC_STATIC_VALUE,6,8,79,9
  EDITTEXT      IDC_EDIT_NAME,94,6,32,12,ES_UPPERCASE | ES_AUTOHSCROLL
  LTEXT         "Prefix",IDC_STATIC,6,25,30,9
  COMBOBOX      IDC_COMBO_PREFIX,94,23,32,36,CBS_DROPDOWN | CBS_SORT |
            WS_VSCROLL | WS_TABSTOP
  DEFPUSHBUTTON   "OK",IDOK,22,40,37,14
  PUSHBUTTON      "Cancel",IDCANCEL,66,40,37,14
END

IDD_DIALOG_OBJECT_NAMES DIALOG DISCARDABLE  0, 0, 141, 61
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Object Property"
FONT 8, "MS Sans Serif"
BEGIN
  LTEXT         "Name Of Object:",IDC_STATIC_VALUE,6,9,79,9
  EDITTEXT      IDC_EDIT_NAME,93,7,39,12,ES_UPPERCASE | ES_AUTOHSCROLL
```

```
  LTEXT          "Name Of Object:",IDC_STATIC_VALUE2,6,25,79,9
  EDITTEXT       IDC_EDIT_NAME2,93,23,39,12,ES_UPPERCASE | ES_AUTOHSCROLL
  DEFPUSHBUTTON  "OK",IDOK,17,43,37,14
  PUSHBUTTON     "Cancel",IDCANCEL,88,43,37,14
END

// Cursor
IDC_PENCIL          CURSOR  DISCARDABLE    "RES\\PENCIL.CUR"

// String Table
STRINGTABLE DISCARDABLE
BEGIN
  IDP_BUSY            "Server is still busy."
  IDP_OLE_INIT_FAILED    "OLE 2.0 initialization failed.  Make sure that the OLE libraries are the
correct version."
  IDP_FAILED_TO_CREATE   "Failed to create object.  Make sure the object is entered in the system
registry."
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
  IDR_MAINFRAME         "PLC Circuit Builder"
  IDR_DRAWCLTYPE         "\nCircuit\nCircuit Document\nCircuit Files
        (*.cct)\n.cct\ncircuitDocument\nCircuitDocument"
  IDS_STRING_COMPLETE_SHUNT "Click on a parent rung to complete this rung"
  IDS_STRING_OBJECT_EXISTS
                  "Component with the same number exists. Use other Number"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
  AFX_IDS_APP_TITLE      "PLC Circuit Builder"
  AFX_IDS_IDLEMESSAGE    "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_INDICATOR_EXT       "EXT"
  ID_INDICATOR_CAPS      "CAP"
  ID_INDICATOR_NUM       "NUM"
  ID_INDICATOR_SCRL      "SCRL"
  ID_INDICATOR_OVR       "OVR"
  ID_INDICATOR_REC       "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_FILE_NEW           "Create a new document"
  ID_FILE_OPEN          "Open an existing document"
  ID_FILE_CLOSE          "Close the active document"
  ID_FILE_SAVE          "Save the active document"
  ID_FILE_SAVE_AS        "Save the active document with a new name"
  ID_FILE_PAGE_SETUP     "Change the printing options"
  ID_FILE_PRINT_SETUP    "Change the printer and printing options"
```

```
  ID_FILE_PRINT          "Print the active document"
  ID_FILE_PRINT_PREVIEW  "Display full pages"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_APP_ABOUT          "Display program information, version number and copyright"
  ID_APP_EXIT           "Quit the application; prompts to save documents"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_FILE_MRU_FILE1     "Open this document"
  ID_FILE_MRU_FILE2     "Open this document"
  ID_FILE_MRU_FILE3     "Open this document"
  ID_FILE_MRU_FILE4     "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_NEXT_PANE          "Switch to the next window pane"
  ID_PREV_PANE          "Switch back to the previous window pane"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_WINDOW_NEW         "Open another window for the active document"
  ID_WINDOW_ARRANGE     "Arrange icons at the bottom of the window"
  ID_WINDOW_CASCADE     "Arrange windows so they overlap"
  ID_WINDOW_TILE_HORZ   "Arrange windows as non-overlapping tiles"
  ID_WINDOW_TILE_VERT   "Arrange windows as non-overlapping tiles"
  ID_WINDOW_SPLIT       "Split the active window into panes"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_EDIT_CLEAR         "Erase the selection"
  ID_EDIT_CLEAR_ALL     "Erase everything"
  ID_EDIT_COPY          "Copy the selection and put it on the Clipboard"
  ID_EDIT_CUT           "Cut the selection and put it on the Clipboard"
  ID_EDIT_FIND          "Find the specified text"
  ID_EDIT_PASTE         "Insert Clipboard contents"
  ID_EDIT_PASTE_LINK    "Insert Clipboard contents and a link to its source"
  ID_EDIT_PASTE_SPECIAL "Insert Clipboard contents with options"
  ID_EDIT_REPEAT        "Repeat the last action"
  ID_EDIT_REPLACE       "Replace specific text with different text"
  ID_EDIT_SELECT_ALL    "Select the entire document"
  ID_EDIT_UNDO          "Undo the last action"
  ID_EDIT_REDO          "Redo the previously undone action"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_VIEW_TOOLBAR       "Show or hide the toolbar"
```

```
  ID_VIEW_STATUS_BAR      "Show or hide the status bar"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_OLE_INSERT_NEW       "Insert new linked or embedded object"
  ID_OLE_EDIT_LINKS       "Edit linked objects"
  ID_OLE_EDIT_CONVERT     "Convert object to different type"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_OLE_VERB_FIRST       "Activate embedded or linked object"
END

STRINGTABLE DISCARDABLE
BEGIN
  AFX_IDS_SCSIZE          "Change the window size"
  AFX_IDS_SCMOVE          "Change the window position"
  AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
  AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
  AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
  AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
  AFX_IDS_SCCLOSE         "Close the active window and prompts to save the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
  AFX_IDS_PREVIEW_CLOSE   "Close print preview mode"
END

STRINGTABLE DISCARDABLE
BEGIN
  AFX_IDS_SCRESTORE       "Restore the window to normal size"
  AFX_IDS_SCTASKLIST      "Activate Task List"
  AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_VIEW_GRID            "Show or hide gridlines"
  ID_DRAW_SELECT          "Choose object selection tool"
  ID_DRAW_LINE            "Choose line drawing tool"
  ID_DRAW_RECT            "Choose rectangle drawing tool"
  ID_DRAW_ROUNDRECT       "Choose rounded rectangle drawing tool"
  ID_DRAW_ELLIPSE         "Choose ellipse drawing tool"
  ID_OBJECT_LINECOLOR     "Change the line or text color"
  ID_OBJECT_FILLCOLOR     "Change the fill or background color"
  ID_OBJECT_MOVETOFRONT   "Move object to front of drawing"
  ID_OBJECT_MOVETOBACK    "Move object to back of drawing"
  ID_OBJECT_MOVEFORWARD   "Move object to next position in z-order"
  ID_OBJECT_MOVEBACK      "Move object to previous position in z-order"
  ID_VIEW_PAPERCOLOR      "Change paper color displayed while editing"
  ID_DRAW_POLYGON         "Choose the polygon drawing tool"
```

```
  ID_VIEW_SHOWOBJECTS    "Toggle OLE object indication"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_EDIT_PROPERTIES     "Edit drawing object properties"
  ID_DRAW_SERIAL_OPEN_SWITCH "Add a new open switch in serial connection"
  IDS_ILLEGAL_OUTPUT_POSITION
                "Output cannot be the first switch on a rung!"
  IDS_STRING_BAD_POINT   "You did not click on a rung."
  IDS_STRING_BACKWARD_POINT "The rung can be completed forward only"
  IDS_INCOMPLETE_RUNG    "Please complete the current rung first."
  ID_DRAW_SERIAL_CLOSED_SWITCH
                "Add a new closed switch in serial connection"
  IDS_STRING_NO_COMPONENT "Select a point left of an object"
  ID_DRAW_RESET_SWITCH   "Add a Reset PLC Switch"
  ID_DRAW_OUTPUT         "Add a PLC Output"
  ID_DRAW_MOVE           "Add Move Instruction"
  ID_DRAW_GREATERTHAN    "Add Greater Than Instruction"
  ID_DRAW_LESSTHAN       "Add Less Than Instruction"
  ID_DRAW_EQUALTO        "Add Equal To Instruction"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_DRAW_ADDITION       "Add Addition Instruction"
  ID_DRAW_SUBTRACTION    "Add Subtraction Instruction"
  ID_DRAW_START          "Add a start element"
  ID_DRAW_STOP           "Add a stop element"
  ID_DRAW_COUNTER        "Add a Counter element"
  ID_DRAW_TIMER          "Add a timer element"
END
#ifndef APSTUDIO_INVOKED
// Generated from the TEXTINCLUDE 3 resource.
#include "res\drawcli.rc2"  // non-App Studio edited resources
#include "afxres.rc"  // Standard components
#include "afxprint.rc"  // printing/print preview resources
#include "afxolecl.rc"  // OLE container resources
#endif   // not APSTUDIO_INVOKED
```