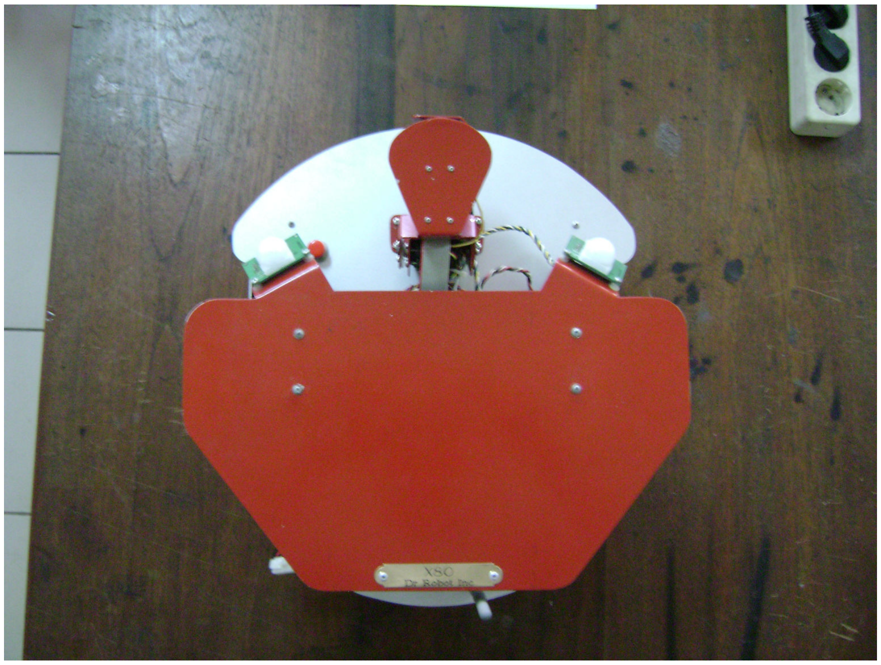
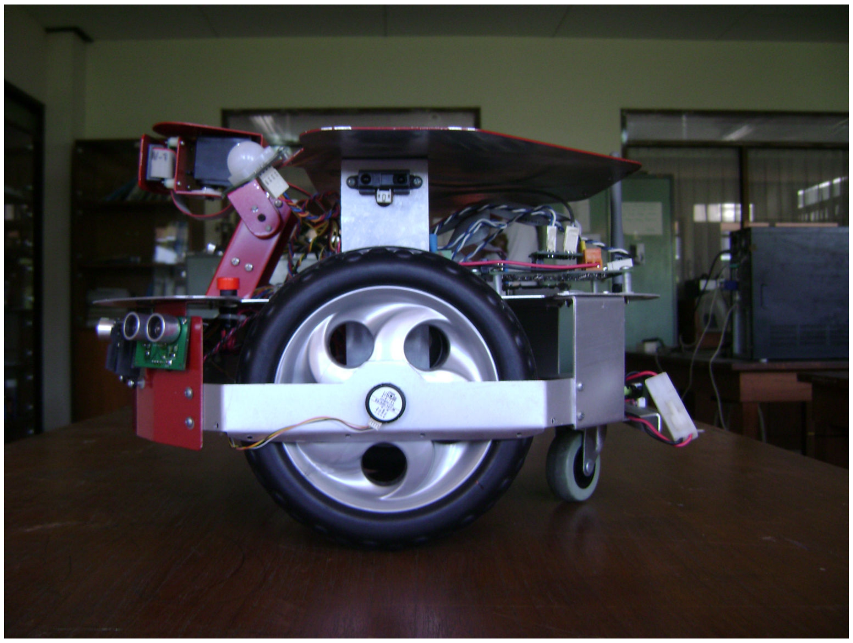
LAMPIRAN A
FOTO WIROBOT X80

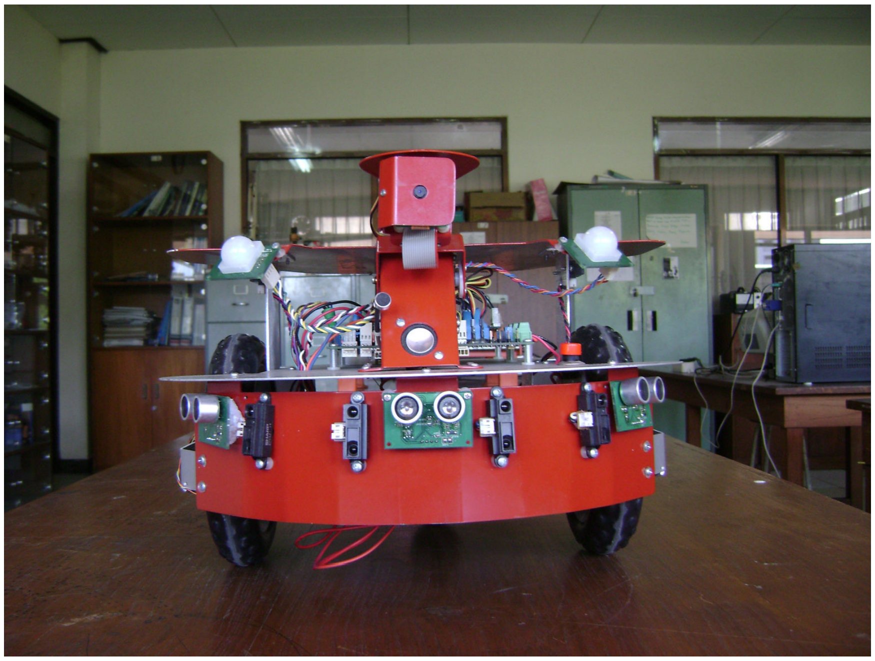
Tampak Atas



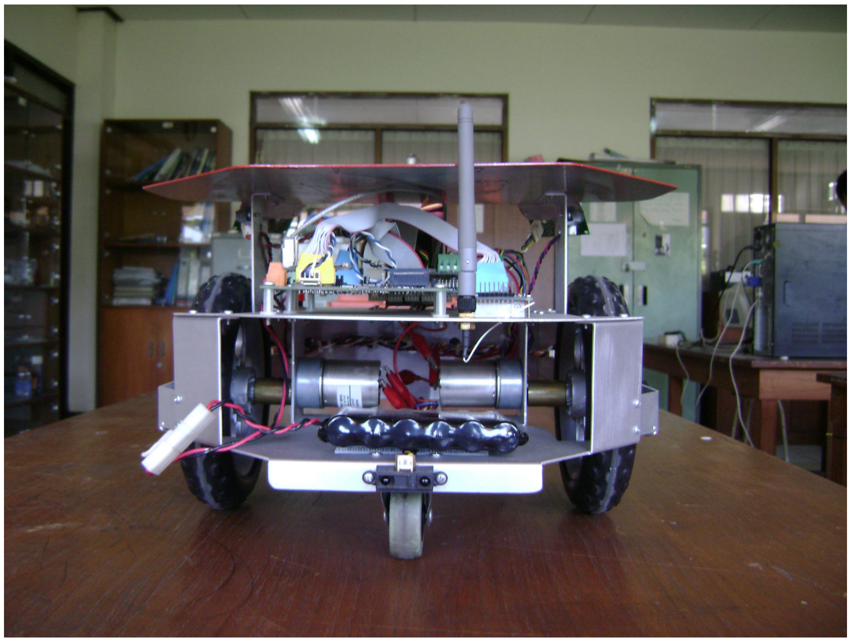
Tampak Samping



Tampak Depan



Tampak Belakang



LAMPIRAN B
PROGRAM PADA PENGONTROL
VISUAL C++

PROGRAM UTAMA

```
// RDPDlg.cpp : implementation file
//

#include "stdafx.h"
#include "RDP.h"
#include "RDPDlg.h"
#include "fstream.h"
#include "Img.h"
#include "time.h"

//#include "BitmapLib.h"

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



#define NO_CONTROL -32768
#define M_PWM 0
#define M_POSITION 1
#define M_VELOCITY 2
#define cFULL_COUNT 32767
#define cWHOLE_RANGE 1200
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

int i;
int a;
class CAboutDlg : public CDialog
{
public:
		CAboutDlg();

// Dialog Data
		//{{AFX_DATA(CAboutDlg)
		enum { IDD = IDD_ABOUTBOX };
		//}}AFX_DATA
```

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

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}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()

/////////////////////////////////////////////////////////////////////////////
// CRDPDlg dialog

//inisialisasi alamat client dan kode warna



//inisialisasi sensor yang di pake pada GUI
```

```
CRDPDlg::CRDPDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CRDPDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CRDPDlg)
	m_Port = 2000;
	m_Text = _T("Dr. Robot");
	m_RoomNum = 0;
	m_IR1 = 0;
	m_IR2 = 0;
	m_IR3 = 0;
	m_IR4 = 0;
	m_IR5 = 0;
	m_IR6 = 0;
	m_IR7 = 0;
	m_Sonar1 = 0;
	m_Sonar2 = 0;
	m_Sonar3 = 0;
	m_PIR1 = 0;
	m_PIR2 = 0;
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

	m_bDlgClose = FALSE;
	m_bRefreshCombo = FALSE;
	InitializeCriticalSection(&m_CriticalSection);
}


CRDPDlg::~CRDPDlg()
{
	DeleteCriticalSection(&m_CriticalSection);
}


//inisialisasi nilai sensor dari robot
void CRDPDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CRDPDlg)
	DDX_Control(pDX, IDC_BUTTON_SERVICE, m_Service);
	DDX_Control(pDX, IDC_DRROBOTSDKCONTROLCTRL1, m_MOTSDK);
	DDX_Control(pDX, IDC_IPADD, m_IPAddr);
	DDX_Control(pDX, IDC_CMB_BIND_TO, m_ComboBindTo);
	DDX_Control(pDX, IDC_CMB_SEND_TO, m_ComboSendTo);
	DDX_Control(pDX, IDC_EDT_OUTPUT, m_Output);
	DDX_Text(pDX, IDC_EDT_PORT, m_Port);
	DDX_Text(pDX, IDC_EDT_SND_TXT, m_Text);
	DDX_Text(pDX, IDC_EDT_ROOM_NUM, m_RoomNum);
	DDX_Text(pDX, IDC_EDT_IR1, m_IR1);
	DDX_Text(pDX, IDC_EDT_IR2, m_IR2);
	DDX_Text(pDX, IDC_EDT_IR3, m_IR3);
	DDX_Text(pDX, IDC_EDT_IR4, m_IR4);
	DDX_Text(pDX, IDC_EDT_IR5, m_IR5);
	DDX_Text(pDX, IDC_EDT_IR6, m_IR6);
	DDX_Text(pDX, IDC_EDT_IR7, m_IR7);
	DDX_Text(pDX, IDC_EDT_SONAR1, m_Sonar1);
	DDX_Text(pDX, IDC_EDT_SONAR2, m_Sonar2);
	DDX_Text(pDX, IDC_EDT_SONAR3, m_Sonar3);
	DDX_Text(pDX, IDC_EDT_PIR1, m_PIR1);
	DDX_Text(pDX, IDC_EDT_PIR2, m_PIR2);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CRDPDlg, CDialog)
	//{{AFX_MSG_MAP(CRDPDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BTN_SERVER, OnBtnServer)
	ON_BN_CLICKED(IDC_BTN_CLOSE_SOCK, OnBtnCloseSock)
	ON_BN_CLICKED(IDC_BTN_CLEAR, OnBtnClear)
	ON_WM_TIMER()
	ON_WM_CLOSE()
	ON_BN_CLICKED(IDC_BTN_CLIENT, OnBtnClient)
	ON_BN_CLICKED(IDC_BTN_SND_TXT, OnBtnSndTxt)
	ON_BN_CLICKED(IDC_BTN_CMD_GET_SRVC, OnBtnCmdGetSrvc)
	ON_BN_CLICKED(IDC_BTN_CMD_DLV_SRVC, OnBtnCmdDlvSrvc)
	ON_BN_CLICKED(IDC_BTN_FWD, OnBtnFwd)
	ON_BN_CLICKED(IDC_BTN_STOP, OnBtnStop)
	ON_BN_CLICKED(IDC_BTN_BACKWARD, OnBtnBackward)
	ON_BN_CLICKED(IDC_BTN_RIGHT, OnBtnRight)
	ON_BN_CLICKED(IDC_BTN_LEFT, OnBtnLeft)
	ON_BN_CLICKED(IDC_BUTTON_BKiS, OnBUTTONBKiS)
	ON_BN_CLICKED(IDC_BUTTON_BKaS, OnBUTTONBKaS)
	ON_BN_CLICKED(IDC_BUTTON_SERVICE, OnButtonService)
	ON_BN_CLICKED(IDC_Btn_TakePhoto, OnBtnTakePhoto)
	ON_BN_CLICKED(IDC_BTN_ServoAtas, OnBTNServoAtas)
	ON_BN_CLICKED(IDC_BTN_ServoKiri, OnBTNServoKiri)
	ON_BN_CLICKED(IDC_BTN_ServoKanan, OnBTNServoKanan)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRDPDlg message handlers

BOOL CRDPDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	SetWindowText(TITLE);

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

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
SetIcon(m_hIcon, FALSE);                    // Set small icon

// TODO: Add extra initialization here
m_MOTSDK.connectRobot("drrobot1");
//m_TISDK.connectRobot("drrobot2");
SetTimer(1, 50, NULL);

// Get all local IP address for each network adapter
DWORD err32 = m_Socket.GetLocalIPs(&m_LocalIP);
if(err32)
        Print("Error retrieving Local IP: %s", GetErrMsg(err32));
else
        m_IPAddr.SetAddress(htonl(m_LocalIP.Get(0)));

CString sText;
#if _UNICODE
        sText = _T("Compiled as UNICODE,");
#else
        sText = _T("Compiled as MBCS,");
#endif

#if _DEBUG
        sText += _T(" DEBUG,");
#else
        sText += _T(" RELEASE,");
#endif

#if PROCESS_EVENTS_IN_GUI_THREAD
        sText += _T(" SingleThreaded,");
#else
        sText += _T(" MultiThreaded");
#endif

sText += _T(" Local IP = ");
if(m_LocalIP.Count() > 1)
        m_ComboBindTo.AddString(_T("All Local IP's"));

for(DWORD i=0; i<m_LocalIP.Count(); i++)
{
        CString sIP = FormatIP(m_LocalIP.Get(i));
        if(i>0)
```

```
				sText += _T(" + ");

			sText += sIP;

			m_ComboBindTo.AddString(sIP);
		}

		m_ComboBindTo.SetCurSel(0);
		Print(sText);

		//refresh the combo box and output  editbox only in GUI thread
		SetTimer(ID_TIMER_UPDATE_GUI, 50, 0);

		//CBitmapLib m_bmap;
		//m_bmap.ChangeColorDepth("Photo.bmp","Photo.bmp",8,NULL);
		Gambar();

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


void CRDPDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
		if ((nID & 0xFFF0) == IDM_ABOUTBOX)
		{
			CAboutDlg dlgAbout;
			dlgAbout.DoModal();
		}
		else
		{
			CDialog::OnSysCommand(nID, lParam);
		}
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon.  For MFC applications using the document/view model,
// this is automatically done for you by the framework.
```

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

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

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

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
		if (m_Picture.status == 0)
		{
			CClientDC dc(this);
			StretchDIBits (dc.m_hDC,
				525,
				370,
				175,
				120,
				0,
				0,
				m_Picture.perolehLebar(),
				m_Picture.perolehTinggi(),
				m_Picture.perolehBit(),
				m_Picture.perolehInfo(),
				DIB_RGB_COLORS,
				SRCCOPY);
		}
	}
```

```
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CRDPDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

BEGIN_EVENTSINK_MAP(CRDPDlg, CDialog)
   //{{AFX_EVENTSINK_MAP(CRDPDlg)
	ON_EVENT(CRDPDlg, IDC_DRROBOTSDKCONTROLCTRL1, 1 /* StandardSensorEvent */,
OnStandardSensorEventDrrobotsdkcontrolctrl1, VTS_NONE)
	ON_EVENT(CRDPDlg, IDC_DRROBOTSDKCONTROLCTRL1, 3 /* CustomSensorEvent */,
OnCustomSensorEventDrrobotsdkcontrolctrl1, VTS_NONE)
	ON_EVENT(CRDPDlg, IDC_DRROBOTSDKCONTROLCTRL1, 2 /* MotorSensorEvent */,
OnMotorSensorEventDrrobotsdkcontrolctrl1, VTS_NONE)
	//}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

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


//tes koneksi server
void CRDPDlg::OnBtnServer()
{
	// TODO: Add your control notification handler code here
	// Switch this application into Server mode and listen for Client connections

	UpdateData(TRUE);		// Load m_Send and m_Port

	if(m_Socket.GetSocketCount())
	{
		Print(_T("Socket already in use!"));
		return;
	}

	DWORD dwBindIP = 0;
	int dSel = m_ComboBindTo.GetCurSel();
	if(dSel > 0)
```

```
		dwBindIP = m_LocalIP.Get(dSel = 1);

	CString sBindIP;
	m_ComboBindTo.GetWindowText(sBindIP);

	DWORD dwErr = m_Socket.Listen(dwBindIP, m_Port);
	if(dwErr)
		Print(_T("Listen Error %s"), GetErrMsg(dwErr));
	else
		Print(_T("Listening (%s) on Port %d... (waiting for FD_ACCEPT)."), sBindIP, m_Port);

	if(dwErr)
	{
		CloseSockets();
		return;
	}

	// runs until an error occurred or all sockets have closed
	#if PROCESS_EVENTS_IN_GUI_THREAD
		ProcessEvents();
	#else
		DWORD dwID;
		m_Thread = CreateThread(0, 0, ProcessEventThread, this, 0, dwID);
	#endif
}


////////////////////CLIENT///////////////////////////////////////CLIENT//////////////////////////////////////////
void CRDPDlg::OnBtnClient()
{
	// TODO: Add your control notification handler code here
	// Switch this application into Client mode and connect to Server

	UpdateData(TRUE);		// Load m_Text and m_Port

	if(m_Socket.GetSocketCount())
	{
		Print(_T("Socket already in use!"));
		return;
	}
```

```
        DWORD dwIP;
        m_IPAddr.GetAddress(dwIP);
        dwIP = htonl(dwIP);

        DWORD dwErr = m_Socket.ConnectTo(dwIP, m_Port);
        if(dwErr)
                Print(_T("Connect Error %s"), GetErrMsg(dwErr));
        else
                Print(_T("Connecting to Server (%s) on Port %d....(waiting for FD_CONNECT)"),
FormatIP(dwIP), m_Port);

        if(dwErr)
        {
                CloseSockets();
                return;
        }

        // runs until an error occured or all sockets have closed
        #if PROCESS_EVENTS_IN_GUI_THREAD
                ProcessEvents();
        #else
                DWORD dwID;
                m_Thread = CreateThread(0, 0, ProcessEventThread, this, 0, &dwID);
        #endif
}

ULONG WINAPI CRDPDlg::ProcessEventThread(void *pParam)
{
        CRDPDlg *pThis = (CRDPDlg*)pParam;
        pThis->ProcessEvents();
        CloseHandle(pThis->m_Thread);
        return 0;
}

void CRDPDlg::ProcessEvents()
{
        // Process all events which occur on one of the open sockets
        BOOL bServer = (m_Socket.GetState() & TCP::cSocket::E_Server);

        if(bServer)
                SetWindowText(TITLE + _T(" - Server"));
        else
                SetWindowText(TITLE + _T(" - Client"));

        //loop run until the main window was closed or a severe error occurred
        while(TRUE)
        {
                #if PROCESS_EVENTS_IN_GUI_THREAD
                        PumpMessages();
                        DWORD dwTimeout = 50;    // 50ms
                #else
                        DWORD dwTimeout = INFINITE;
                #endif

                char cBuf[512];      // should be use a bigger buffer!

                SOCKET h_Socket;
                DWORD dwEvent, dwIP, dwRead, dwSent;
                DWORD dwErr = m_Socket.ProcessEvents(dwTimeout, &dwEvent, &dwIP, &h_Socket,

cBuf, sizeof(cBuf)-2, &dwRead, &dwSent);

                // Main dialog was closed -> !Immediately! stop all output and printing into GUI
                // otherwise the application will not shut down correctly and the EXE keep running.
                // (only visible in Task Manager).
                // There may appear a lot of other strange things when the Events thread still runs
                // while the GUI thread already finished!.
                if(m_bDlgClose)
                        return;                 // return not Break!

                if(dwErr == ERROR_TIMEOUT)        // 50ms interval has elapsed
                        continue;

                CString sMsg, sEvent;
                if(dwEvent)                     // ATTENTION: dwEvent may be == 0 -> do nothing.
                {
                        if(bServer)
                                sEvent.Format(_T("Client %X (%s) --> "), h_Socket,
FormatIP(dwIP));
                        else
                                sEvent.Format(_T("Server (%s) --> "), FormatIP(dwIP));
```

```
                    if(dwEvent & FD_ACCEPT)                                    sEvent
+= _T("FD_ACCEPT ");
                    if(dwEvent & FD_CONNECT)            sEvent += _T("FD_CONNECT ");
                    if(dwEvent & FD_CLOSE)              sEvent += _T("FD_CLOSE ");
                    if(dwEvent & FD_READ)               sEvent += _T("FD_READ ");
                    if(dwEvent & FD_WRITE)              sEvent += _T("FD_WRITE ");
                    if(dwEvent & FD_OOB)                sEvent += _T("FD_OOB ");
                    if(dwEvent & FD_QOS)                sEvent += _T("FD_QOS ");
                    if(dwEvent & FD_GROUP_QOS)              sEvent += _T("FD_GROUP_QOS
");
                    if(dwEvent & FD_ROUTING_INTERFACE_CHANGE) sEvent +=
_T("FD_ROUTING_INTERFACE_CHANGE ");
                    if(dwEvent & FD_ADDRESS_LIST_CHANGE)    sEvent +=
_T("FD_ADDRESS_LIST_CHANGE ");

                    if(dwEvent & FD_READ)
                    {
                          // Append terminating unicode Zero! (Unicode: 1 character = 2 Bytes)
                          cBuf[dwRead]   = 0;
                          cBuf[dwRead+1] = 0;

                          sMsg.Format(_T(", %d Bytes: '%s'"), dwRead, cBuf);        //
sOMETHING MUST BE....

                          if(strncmp(cBuf,"CMD",3) == 0)
                          {
                                // Server receive CMD command from client
                                CString sClientCmd;

                                sClientCmd = cBuf;
                                sClientCmd.Delete(0, 4);
                                if(sClientCmd.Compare("MOVE_FORWARD") == 0)
                                {

                                      OnBtnFwd();
                                }
                                else if(sClientCmd.Compare("MOVE_LEFT") == 0)
                                {

                                      OnBtnLeft();
                                }
```

```
                                        else if(sClientCmd.Compare("MOVE_RIGHT") == 0)
                                        {

                                                OnBtnRight();
                                        }
                                        else if(sClientCmd.Compare("MOVE_BACK") == 0)
                                        {

                                                OnBtnBackward();
                                        }

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

                    else if(sClientCmd.Compare("NEED_SERVICE") == 0)
                                        {

                                                JlnLurus();
                                        }




                                        else if(sClientCmd.Compare("DELIVER_SERVICE") ==
0)

                                        {
                                                Base();
                                                JlnLurus();
                                        }
                                        else
                                        {

                                                Base();
                                        }
                                }

                        }

                        if(dwEvent & FD_WRITE)
                        {
                                sMsg.Format(_T(", %d Bytes Sent"), dwSent);
                        }
                        Print(_T("%s"), sEvent + sMsg);
                        IPclient = FormatIP(dwIP);
                }

                // It is not necessary to update the Combobox after every FD_READ
                m_bRefreshCombo |= (dwEvent & (FD_ACCEPT | FD_CONNECT | FD_CLOSE) ||
dwErr);

                if(dwErr)
                {
                        // m_Socket.Close() has been called -> don't print this error message
                        if(dwErr == WSAENOTCONN)
                                break;

                        // print all the other error messages
                        Print(_T("ProcessEvent Error %s"), GetErrMsg(dwErr));

                        // An error normaly means that the socket has a problem -> abort the loop
                        // a few error should not abort the processing!
                        if(dwErr != WSAECONNABORTED &&          // after the other side was
killed in Task Manager
                      dwErr != WSAECONNRESET &&   // Connection reset by peer
                      dwErr != WSAECONNREFUSED)   // FD_ACCEPT with already 62 clients
connected
                      break;
                }
          };          // end loop

          CloseSockets();

          SetWindowText(TITLE);
          if(bServer)
                Print(_T("Stop Listening.\r\n"));
          else
                Print(_T("Connection Abandoned.\r\n"));
}

void CRDPDlg::OnBtnCloseSock()
{
          // TODO: Add your control notification handler code here
```

```
        // Close all open Socket
        if(!m_Socket.GetSocketCount())
                Print(_T("No Socket Open!"));
        else
                CloseSockets();
}

void CRDPDlg::CloseSockets()
{
        // Close all open socket (if any)
        if(m_Socket.GetSocketCount())
        {
                m_Socket.Close();
                Print(_T("Socket(s) Closed."));

                m_ComboSendTo.ResetContent();
        }
}

void CRDPDlg::OnBtnClear()
{
        // TODO: Add your control notification handler code here
        m_Output.SetWindowText(_T(""));
}

void CRDPDlg::OnTimer(UINT nIDEvent)
{
        // TODO: Add your message handler code here and/or call default
        m_MOTSDK.TakePhoto ();
        CDialog::OnTimer(nIDEvent);
        //CDialog::OnTimer(nIDEvent);
        if(nIDEvent != ID_TIMER_UPDATE_GUI)
                return;

        // Update Combobox
        if(m_bRefreshCombo)
        {
                m_bRefreshCombo = FALSE;

                int dSel = m_ComboSendTo.GetCurSel();
                m_ComboSendTo.ResetContent();

                DWORD dwCount = m_Socket.GetSocketCount();
                if(m_Socket.GetState() & TCP::cSocket::E_Connected)
                {
                        for(DWORD i=0; i<dwCount; i++)
                        {
                                m_ComboSendTo.AddString(FormatDisplayName(i));
                        }

                        // maintain the current sellection if possible
                        m_ComboSendTo.SetCurSel(max(0, min((int) dwCount-1, dSel)));
                }
        }

        // Update Output Editbox
        // the variable m_sOutput is manipulated from 2 threads
        // the critical section assures thread safety

        EnterCriticalSection(&m_CriticalSection);
        CString sAppend = m_sOutput;
        m_sOutput.Empty();
        LeaveCriticalSection(&m_CriticalSection);

        if(sAppend.GetLength())
        {
                CString sText;
                m_Output.GetWindowText(sText);
                sText += sAppend;

                m_Output.SetWindowText(sText);

                // scroll to the last line
                m_Output.SetSel(sText.GetLength(), sText.GetLength());
        }
}

void CRDPDlg::PumpMessages()
{
        // Allow to update GUI from within an endless loop without needing an extra thread
        MSG kMsg;
        while(PeekMessage(&kMsg, NULL, NULL, NULL, PM_NOREMOVE))
        {
                AfxGetThread()->PumpMessage();
        }
}

void CRDPDlg::OnClose()
{
        // TODO: Add your message handler code here and/or call default
        // When the main dialog is closed: set the m_bDlgClose flag to abort the ProcessEvent() thread!
        m_bDlgClose = TRUE;
        m_Socket.Close();
        CDialog::OnClose();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////////
// HELPER
//////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////////
void CRDPDlg::Print(CString sFormat, ...)
{
        // Append formatted text to the string m_sOutput which is later written to the output
        // Editbox in the GUI thread
        const DWORD BUFLEN = 5000;

        TCHAR tLine[BUFLEN + 1];

        va_list args;
        va_start(args, sFormat);

        _vsntprintf(tLine, BUFLEN, sFormat, args);

        // if new line should be longer than 5000 characters it will be cropped.
        tLine[BUFLEN] = 0;

        // m_sOutput variable is manipulated from two threads
        // the critical section assures thread safety
        EnterCriticalSection(&m_CriticalSection);

        m_sOutput += tLine;
        m_sOutput += _T("\r\n");

        LeaveCriticalSection(&m_CriticalSection);
}

CString CRDPDlg::FormatDisplayName(DWORD dwIndex)
{
        // Format the string to displayed in the Combobox and the output box
        // returns "Server (192.168.1.100)" or "Client 71C (192.168.1.100)"
        SOCKET h_Socket;
        DWORD dwIP;
        m_Socket.GetSocket(dwIndex, &h_Socket, &dwIP);

        CString sDisp;
        if(dwIndex == 0)    // Combobox index 0 is used on the server to send to all clients
        {
                if(m_Socket.GetState() & TCP::cSocket::E_Server)
                        sDisp.Format(_T("All Clients"));
                else
                        sDisp.Format(_T("Server (%s)"), FormatIP(dwIP));
        }
        else
        {
                sDisp.Format(_T("Client %X (%s)"), h_Socket, FormatIP(dwIP));
        }

        return sDisp;
}

CString CRDPDlg::FormatIP(DWORD dwIP)
{
        // Format an IP address "192.168.0.100"
        BYTE* pAddr = (BYTE*)&dwIP;

        CString sIP;
        sIP.Format(_T("%d.%d.%d.%d"), pAddr[0], pAddr[1], pAddr[2], pAddr[3]);

        return sIP;
}

CString CRDPDlg::GetErrMsg(DWORD dwError)
```

```
{
        // Get a human readable error message for an API error code
        // Some translation of error codes are really stupid --> show the original error code
        CString sCode;
        switch(dwError)
        {
                case WSAEINTR:                          sCode = _T("WSAEINTR");
break;
                case WSAEBADF:                          sCode = _T("WSAEBADF");
break;
                case WSAEACCES:                                 sCode =
_T("WSAEACCES"); break;
                case WSAEFAULT:                                 sCode =
_T("WSAEFAULT"); break;
                case WSAEINVAL:                                 sCode =
_T("WSAEINVAL"); break;
                case WSAEMFILE:                                 sCode =
_T("WSAEMFILE"); break;
                case WSAEWOULDBLOCK:                    sCode =
_T("WSAEWOULDBLOCK"); break;
                case WSAEINPROGRESS:            sCode = _T("WSAEINPROGRESS");
break;
                case WSAEALREADY:                       sCode =
_T("WSAEALREADY"); break;
                case WSAENOTSOCK:                       sCode =
_T("WSAENOTSOCK"); break;
                case WSAEDESTADDRREQ:                   sCode =
_T("WSAEDESTADDRREQ"); break;
                case WSAEMSGSIZE:                       sCode =
_T("WSAEMSGSIZE"); break;
                case WSAEPROTOTYPE:                     sCode =
_T("WSAEPROTOTYPE"); break;
                case WSAENOPROTOOPT:            sCode = _T("WSAENOPROTOOPT");
break;
                case WSAEPROTONOSUPPORT:         sCode =
_T("WSAEPROTONOSUPPORT"); break;
                case WSAESOCKTNOSUPPORT:         sCode =
_T("WSAESOCKTNOSUPPORT"); break;
                case WSAEOPNOTSUPP:                     sCode =
_T("WSAEOPNOTSUPP"); break;
```

```
			case WSAEPFNOSUPPORT:				sCode =
_T("WSAEPFNOSUPPORT"); break;
			case WSAEAFNOSUPPORT:				sCode =
_T("WSAEAFNOSUPPORT"); break;
			case WSAEADDRINUSE:				sCode =
_T("WSAEADDRINUSE"); break;
			case WSAEADDRNOTAVAIL:				sCode =
_T("WSAEADDRNOTAVAIL"); break;
			case WSAENETDOWN:				sCode =
_T("WSAENETDOWN"); break;
			case WSAENETUNREACH:			sCode = _T("WSAENETUNREACH");
break;
			case WSAENETRESET:				sCode =
_T("WSAENETRESET"); break;
			case WSAECONNABORTED:				sCode =
_T("WSAECONNABORTED"); break;
			case WSAECONNRESET:				sCode =
_T("WSAECONNRESET"); break;
			case WSAENOBUFS:				sCode =
_T("WSAENOBUFS"); break;
			case WSAEISCONN:				sCode =
_T("WSAEISCONN"); break;
			case WSAENOTCONN:				sCode =
_T("WSAENOTCONN"); break;
			case WSAESHUTDOWN:				sCode =
_T("WSAESHUTDOWN"); break;
			case WSAETOOMANYREFS:				sCode =
_T("WSAETOOMANYREFS"); break;
			case WSAETIMEDOUT:				sCode =
_T("WSAETIMEDOUT"); break;
			case WSAECONNREFUSED:				sCode =
_T("WSAECONNREFUSED"); break;
			case WSAELOOP:					sCode = _T("WSAELOOP");
break;
			case WSAENAMETOOLONG:				sCode =
_T("WSAENAMETOOLONG"); break;
			case WSAEHOSTDOWN:				sCode =
_T("WSAEHOSTDOWN"); break;
			case WSAEHOSTUNREACH:				sCode =
_T("WSAEHOSTUNREACH"); break;
```

```
			case WSAENOTEMPTY:				sCode =
_T("WSAENOTEMPTY"); break;
			case WSAEPROCLIM:				sCode =
_T("WSAEPROCLIM"); break;
			case WSAEUSERS:					sCode =
_T("WSAEUSERS"); break;
			case WSAEDQUOT:					sCode =
_T("WSAEDQUOT"); break;
			case WSAESTALE:					sCode =
_T("WSAESTALE"); break;
			case WSAEREMOTE:				sCode =
_T("WSAEREMOTE"); break;
			case WSASYSNOTREADY:			sCode = _T("WSASYSNOTREADY");
break;
			case WSAVERNOTSUPPORTED:		sCode =
_T("WSAVERNOTSUPPORTED"); break;
			case WSANOTINITIALISED:				sCode =
_T("WSANOTINITIALISED"); break;
			case WSAEDISCON:				sCode =
_T("WSAEDISCON"); break;
			case WSAENOMORE:				sCode =
_T("WSAENOMORE"); break;
			case WSAECANCELLED:				sCode =
_T("WSAECANCELLED"); break;
			case WSAEINVALIDPROCTABLE:		sCode =
_T("WSAEINVALIDPROCTABLE"); break;
			case WSAEINVALIDPROVIDER:		sCode =
_T("WSAEINVALIDPROVIDER"); break;
			case WSAEPROVIDERFAILEDINIT:	sCode = _T("WSAEPROVIDERFAILEDINIT");
break;
			case WSASYSCALLFAILURE:				sCode =
_T("WSASYSCALLFAILURE"); break;
			case WSASERVICE_NOT_FOUND:		sCode =
_T("WSASERVICE_NOT_FOUND"); break;
			case WSATYPE_NOT_FOUND:				sCode =
_T("WSATYPE_NOT_FOUND"); break;
			case WSA_E_NO_MORE:				sCode =
_T("WSA_E_NO_MORE"); break;
			case WSA_E_CANCELLED:				sCode =
_T("WSA_E_CANCELLED"); break;
```

```
                    case WSAEREFUSED:                               sCode =
_T("WSAEREFUSED"); break;
                    case WSAHOST_NOT_FOUND:                         sCode =
_T("WSAHOST_NOT_FOUND"); break;
                    case WSATRY_AGAIN:                              sCode =
_T("WSATRY_AGAIN"); break;
                    case WSANO_RECOVERY:                  sCode = _T("WSANO_RECOVERY");
break;
                    case WSANO_DATA:                                sCode =
_T("WSANO_DATA"); break;
                    case WSA_IO_PENDING:                  sCode = _T("WSA_IO_PENDING");
break;
                    case WSA_IO_INCOMPLETE:                         sCode =
_T("WSA_IO_INCOMPLETE"); break;
                    case WSA_INVALID_HANDLE:              sCode =
_T("WSA_INVALID_HANDLE"); break;
                    case WSA_INVALID_PARAMETER:           sCode =
_T("WSA_INVALID_PARAMETER"); break;
                    case WSA_NOT_ENOUGH_MEMORY:                     sCode =
_T("WSA_NOT_ENOUGH_MEMORY"); break;
                    case WSA_OPERATION_ABORTED:           sCode =
_T("WSA_OPERATION_ABORTED"); break;
                    default:
                              sCode.Format(_T("Code %u"), dwError);
                              break;
          }

          CString sOut;
          const DWORD BUFLEN = 1000;
          TCHAR tBuf[BUFLEN];

          if(FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0, dwError, 0, tBuf, BUFLEN, 0))
                    sOut.Format(_T("%s : %s"), sCode, tBuf);
          else
                    sOut.Format(_T("%s : Windows has no explanation for this error"), sCode);

          sOut.TrimRight();   // some messages end with useless Linefeeds
          return sOut;
}
```

```
void CRDPDlg::OnBtnSndTxt()
{
        // TODO: Add your control notification handler code here
        // Send a text string to one or multiple destinations

        UpdateData(TRUE);           // Load m_Text and m_Port

        if(!m_ComboSendTo.GetCount())
        {
                Print(_T("Not Connected"));
                return;
        }

        #if SEND_LARGE_DATA > 0
                // SEND_LARGE_DATA = 10000 -> send a 100 Kbyte string "AAAAA...", each time
with another character
                static TCHAR tChar = 'A';
                CString sSendData(tChar++, SEND_LARGE_DATA/sizeof (TCHAR));
                if(tChar > 'Z') tChar = 'A';
        #else
                // send the string that the user has entered
                CString sSendData = m_Text;
        #endif

        if(!sSendData.GetLength())
        {
                Print(_T("Error, You must enter a Text!"));
                return;
        }

        int dSel = m_ComboSendTo.GetCurSel();

        // Combobox index=0 on server -> send to all connected clients
        if(dSel == 0 && (m_Socket.GetState() & TCP::cSocket::E_Server))
        {
                // Socket[0] is not connected on the server!!
                for(DWORD i=1; i<m_Socket.GetSocketCount(); i++)
                {
                        if(!SendTo(i, sSendData))
                                break;
```

```
			}
		}
		else
		{
			SendTo(dSel, sSendData);
		}
}

BOOL CRDPDlg::SendTo(DWORD dwIndex, CString sSendData)
{
		// Sends data to the socket with the given index
		// A "\r\n" in the input string is replaced with the linebreak
		// returns FALSE when the socket have been closed due to a severe error

		CString sText = sSendData;
		if(sText.GetLength() > 50)
			sText = sText.Left(50) + "...<cut>";

		sSendData.Replace(_T("\\n"), _T("\n"));
		sSendData.Replace(_T("\\r"), _T("\r"));

		//if unicode: 1 character = 2 bytes!
		DWORD dwLen = sSendData.GetLength() * sizeof(TCHAR);
		char *pData = (char*)(const TCHAR*)sSendData;

		Print(_T("Sending %d Bytes to %s: '%s'"), dwLen, FormatDisplayName(dwIndex), sText);

		SOCKET hSocket;
		m_Socket.GetSocket(dwIndex, &hSocket, 0);
		DWORD dwErr = m_Socket.SendTo(hSocket, pData, dwLen);

		switch(dwErr)
		{
		case 0:
			return TRUE;
		case WSAEWOULDBLOCK:
			Print(_T("WSAEWOULDBLOCK -> The data will be sent after the next FD_WRITE
event."));
			return TRUE;
		case WSA_IO_PENDING:
```

```
                        Print(_T("WSA_IO_PENDING -> Error: A previous send operation is still pending. This
data will not be sent"));
                        return TRUE;
            default:
                        Print(_T("%s"), _T(" -> Error ") + GetErrMsg(dwErr));
                        CloseSockets();
                        return FALSE;
            };
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Robot API
////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////

void CRDPDlg::OnStandardSensorEventDrrobotsdkcontrolctrl1()
{
            // TODO: Add your control notification handler code here
            m_Sonar1 = m_MOTSDK.GetSensorSonar1();
            m_Sonar2 = m_MOTSDK.GetSensorSonar2();
            m_Sonar3 = m_MOTSDK.GetSensorSonar3();

            m_IR1 = m_MOTSDK.GetSensorIRRange();



            m_PIR1 = m_MOTSDK.GetSensorHumanMotion1();
            m_PIR2 = m_MOTSDK.GetSensorHumanMotion2();

            UpdateData(false);
}

void CRDPDlg::OnCustomSensorEventDrrobotsdkcontrolctrl1()
{
            // TODO: Add your control notification handler code here
            m_IR2 = m_MOTSDK.GetCustomAD3();
            m_IR3 = m_MOTSDK.GetCustomAD4();
            m_IR4 = m_MOTSDK.GetCustomAD5();
            m_IR5 = m_MOTSDK.GetCustomAD6();
            m_IR6 = m_MOTSDK.GetCustomAD7();
            m_IR7 = m_MOTSDK.GetCustomAD8();
```

```
	//m_PIR1 = m_MOTSDK.GetSensorHumanMotion1();
	//m_PIR2 = m_MOTSDK.GetSensorHumanMotion2();

	UpdateData(false);
}


void CRDPDlg::OnMotorSensorEventDrrobotsdkcontrolctrl1()
{
	// TODO: Add your control notification handler code here
	m_encoder1 = m_MOTSDK.GetEncoderPulse1();
	m_encoder2 = m_MOTSDK.GetEncoderPulse2();
	UpdateData(false);
}


/////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////////////
// Client Commands
/////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////////////

void CRDPDlg::OnBtnCmdGetSrvc()
{
	// TODO: Add your control notification handler code here
	// Send Client command to server for getting some service from Robot
	// Command: CMD NEED_SERVICE

	CString sCmd = "CMD NEED_SERVICE";
	SendClientCmd(sCmd);
}

void CRDPDlg::OnBtnCmdDlvSrvc()
{
	// TODO: Add your control notification handler code here
	// Command: CMD DELIVER_SERVICE

	CString sCmd = "CMD DELIVER_SERVICE";
	SendClientCmd(sCmd);
}
```

```
void CRDPDlg::SendClientCmd(CString sClientCmd)
{
        int dSel = m_ComboSendTo.GetCurSel();
        if(dSel == 0 && (m_Socket.GetState() & TCP::cSocket::E_Server))
        {
                Print(_T("Not a Client request!"));
        }
        else
        {
                SendTo(dSel, sClientCmd);
        }
}



//Program Buat Kerja Robot

void CRDPDlg::MoveForward()
{
        m_MOTSDK.SetDcMotorControlMode (0,M_VELOCITY);
        m_MOTSDK.SetDcMotorControlMode (1,M_VELOCITY);
        //m_MOTSDK.DcMotorPwmTimeCtrAll(6668,26077,-32768,-32768,-32768,-32768,800);
        m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
        m_MOTSDK.SetDcMotorVelocityControlPID (1, 30, 10, 0);
        m_MOTSDK.DcMotorVelocityNonTimeCtrAll (-200,
200,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL);

}


void CRDPDlg::JlnLurus()

{

        if(m_Sonar3 < 20)
        {
                OnBtnStop();
                OnMotorSensorEventDrrobotsdkcontrolctrl1();
                belokkirisedikit();
                OnBtnFwd();
```

```
		}

		if(m_Sonar3 > 35)
		{
			OnBtnStop();
			OnMotorSensorEventDrrobotsdkcontrolctrl1();
			belokkanansedikit();
			OnBtnFwd();
		}

		if(m_Sonar3 > 20 && m_Sonar3 < 35)
			OnBtnFwd();

		if(m_Sonar3 == 255)
		{	OnBtnStop();
			OnBTNServoAtas();
			OnBTNServoKanan();
			OnBtnTakePhoto();

			if(WarnaDominan == warna)

			{
				SisaPintu();
				OnBtnStop();
				OnMotorSensorEventDrrobotsdkcontrolctrl1();
				OnBtnRight();
				MasukRuangan();
				CariManusia();
			}

		}


	}



void CRDPDlg::BelokKanan()
{
```

```
                        long cmd1,cmd2;

    cmd1 = m_encoder1 - cWHOLE_RANGE / 3;
    cmd2 = m_encoder2 - cWHOLE_RANGE / 3;

    // change cmd1, cmd2 to valid data range
    if (cmd1 < 0) cmd1 = cmd1 + cFULL_COUNT;
    if (cmd2 < 0) cmd2 = cmd2 + cFULL_COUNT;
    if (cmd1 > cFULL_COUNT) cmd1 = cmd1 - cFULL_COUNT;
    if (cmd2 > cFULL_COUNT) cmd2 = cmd2 - cFULL_COUNT;


            m_MOTSDK.SetDcMotorControlMode (0,M_POSITION);
            m_MOTSDK.SetDcMotorControlMode (1,M_POSITION);
            m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
            m_MOTSDK.SetDcMotorPositionControlPID (0, 600,30,600);
            m_MOTSDK.SetDcMotorPositionControlPID (1, 600,30,600);
            m_MOTSDK.DcMotorPositionTimeCtrAll
(cmd1,cmd2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL,1000);
   Sleep(1000);
}


void CRDPDlg::BelokKiri()
{
            long cmd1,cmd2;

    cmd1 = m_encoder1 + cWHOLE_RANGE / 3;
    cmd2 = m_encoder2 + cWHOLE_RANGE / 3;

   //change cmd1, cmd2 to valid data range
    if (cmd1 < 0) cmd1 = cmd1 + cFULL_COUNT;
    if (cmd2 < 0) cmd2 = cmd2 + cFULL_COUNT;
    if (cmd1 > cFULL_COUNT) cmd1 = cmd1 - cFULL_COUNT;
    if (cmd2 > cFULL_COUNT) cmd2 = cmd2 - cFULL_COUNT;


            m_MOTSDK.SetDcMotorControlMode (0,M_POSITION);
            m_MOTSDK.SetDcMotorControlMode (1,M_POSITION);
            m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
            m_MOTSDK.SetDcMotorPositionControlPID (0, 600,30,600);
```

```
        m_MOTSDK.SetDcMotorPositionControlPID (1, 600,30,600);
        m_MOTSDK.DcMotorPositionTimeCtrAll
(cmd1,cmd2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL,1000);
}



void CRDPDlg::OnBtnFwd()
{
        // TODO: Add your control notification handler code here
        m_MOTSDK.SetDcMotorControlMode (0,M_VELOCITY);
        m_MOTSDK.SetDcMotorControlMode (1,M_VELOCITY);
        m_MOTSDK.DcMotorPwmTimeCtrAll(6000,26000,-32768,-32768,-32768,-32768,800);
        m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
        m_MOTSDK.SetDcMotorVelocityControlPID (1, 40, 10, 0);
        m_MOTSDK.DcMotorVelocityNonTimeCtrAll (-400,
400,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL);

}

void CRDPDlg::OnBtnStop()
{
        // TODO: Add your control notification handler code here
        m_MOTSDK.SuspendDcMotor(0);
        m_MOTSDK.SuspendDcMotor(1);

}


void CRDPDlg::OnBtnBackward()
{
        // TODO: Add your control notification handler code here
        m_MOTSDK.SetDcMotorControlMode (1,0);
        m_MOTSDK.SetDcMotorControlMode (0,0);
        m_MOTSDK.DcMotorPwmTimeCtrAll(26077,6668,-32768,-32768,-32768,-32768,800);
        m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
        m_MOTSDK.SetDcMotorVelocityControlPID (1, 30, 10, 0);
        m_MOTSDK.DcMotorVelocityNonTimeCtrAll (300, -
300,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL);
}
```

```
void CRDPDlg::OnBtnRight()
{
	// TODO: Add your control notification handler code here
	BelokKanan();

}


void CRDPDlg::OnBtnLeft()
{
	// TODO: Add your control notification handler code here
	BelokKiri();

}




void CRDPDlg::Gambar()
{
	m_Picture.muatBerkas ("Photo.bmp");
	if (m_Picture.status != 0)
	{
		MessageBox("Berkas tak dapat dibuka");
		return;
	}

	Invalidate();
}

void CRDPDlg::maju()
{
	OnBtnFwd();
	Sleep(5000);
	OnBtnStop();

}

void CRDPDlg::belokkanansedikit()
{
```

```
            long cmd1,cmd2;

   cmd1 = m_encoder1 - cWHOLE_RANGE / 12;
   cmd2 = m_encoder2 - cWHOLE_RANGE / 12;

   // change cmd1, cmd2 to valid data range
   if (cmd1 < 0) cmd1 = cmd1 + cFULL_COUNT;
   if (cmd2 < 0) cmd2 = cmd2 + cFULL_COUNT;
   if (cmd1 > cFULL_COUNT) cmd1 = cmd1 - cFULL_COUNT;
   if (cmd2 > cFULL_COUNT) cmd2 = cmd2 - cFULL_COUNT;


            m_MOTSDK.SetDcMotorControlMode (0,M_POSITION);
            m_MOTSDK.SetDcMotorControlMode (1,M_POSITION);
            m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
            m_MOTSDK.SetDcMotorPositionControlPID (0, 600,30,600);
            m_MOTSDK.SetDcMotorPositionControlPID (1, 600,30,600);
            m_MOTSDK.DcMotorPositionTimeCtrAll
(cmd1,cmd2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL,1000);

}

void CRDPDlg::belokkirisedikit()
{
            long cmd1,cmd2;

   cmd1 = m_encoder1 + cWHOLE_RANGE / 12;
   cmd2 = m_encoder2 + cWHOLE_RANGE / 12;

  //change cmd1, cmd2 to valid data range
   if (cmd1 < 0) cmd1 = cmd1 + cFULL_COUNT;
   if (cmd2 < 0) cmd2 = cmd2 + cFULL_COUNT;
   if (cmd1 > cFULL_COUNT) cmd1 = cmd1 - cFULL_COUNT;
   if (cmd2 > cFULL_COUNT) cmd2 = cmd2 - cFULL_COUNT;


            m_MOTSDK.SetDcMotorControlMode (0,M_POSITION);
            m_MOTSDK.SetDcMotorControlMode (1,M_POSITION);
            m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
            m_MOTSDK.SetDcMotorPositionControlPID (0, 600,30,600);
            m_MOTSDK.SetDcMotorPositionControlPID (1, 600,30,600);
            m_MOTSDK.DcMotorPositionTimeCtrAll
(cmd1,cmd2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL,1000);

}

void CRDPDlg::OnBUTTONBKiS()
{
            // TODO: Add your control notification handler code here
            long cmd1,cmd2;

   cmd1 = m_encoder1 + cWHOLE_RANGE / 10;
   cmd2 = m_encoder2 + cWHOLE_RANGE / 10;

  //change cmd1, cmd2 to valid data range
   if (cmd1 < 0) cmd1 = cmd1 + cFULL_COUNT;
   if (cmd2 < 0) cmd2 = cmd2 + cFULL_COUNT;
   if (cmd1 > cFULL_COUNT) cmd1 = cmd1 - cFULL_COUNT;
   if (cmd2 > cFULL_COUNT) cmd2 = cmd2 - cFULL_COUNT;


            m_MOTSDK.SetDcMotorControlMode (0,M_POSITION);
            m_MOTSDK.SetDcMotorControlMode (1,M_POSITION);
            m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
            m_MOTSDK.SetDcMotorPositionControlPID (0, 600,30,600);
            m_MOTSDK.SetDcMotorPositionControlPID (1, 600,30,600);
            m_MOTSDK.DcMotorPositionTimeCtrAll
(cmd1,cmd2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL,1000);
}

void CRDPDlg::OnBUTTONBKaS()
{
            // TODO: Add your control notification handler code here
            long cmd1,cmd2;

   cmd1 = m_encoder1 - cWHOLE_RANGE / 10;
   cmd2 = m_encoder2 - cWHOLE_RANGE / 10;

   // change cmd1, cmd2 to valid data range
   if (cmd1 < 0) cmd1 = cmd1 + cFULL_COUNT;
   if (cmd2 < 0) cmd2 = cmd2 + cFULL_COUNT;
   if (cmd1 > cFULL_COUNT) cmd1 = cmd1 - cFULL_COUNT;
```

```
    if (cmd2 > cFULL_COUNT) cmd2 = cmd2 - cFULL_COUNT;


            m_MOTSDK.SetDcMotorControlMode (0,M_POSITION);
            m_MOTSDK.SetDcMotorControlMode (1,M_POSITION);
            m_MOTSDK.SetDcMotorVelocityControlPID (0, 30, 10, 0);
            m_MOTSDK.SetDcMotorPositionControlPID (0, 600,30,600);
            m_MOTSDK.SetDcMotorPositionControlPID (1, 600,30,600);
            m_MOTSDK.DcMotorPositionTimeCtrAll
(cmd1,cmd2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL,1000);
}


void CRDPDlg::OnButtonService()
{
            // TODO: Add your control notification handler code



}


void CRDPDlg::OnBtnTakePhoto()
{
            // TODO: Add your control notification handler code here
            m_MOTSDK.TakePhoto();
            m_MOTSDK.SavePhotoAsBMP("Photo.bmp");
            bool bVal;
            bVal = m_MOTSDK.SavePhotoAsBMP("Photo.bmp");

            if(bVal==true)
            {
                        Gambar();
            }

}


void CRDPDlg::OnBTNServoAtas()
{
            // TODO: Add your control notification handler code here
            m_MOTSDK.EnableServo (0);
            m_MOTSDK.ServoTimeCtr (0,3500,1500);
}





void CRDPDlg::OnBTNServoKiri()
{
            // TODO: Add your control notification handler code here
            m_MOTSDK.EnableServo (1);

  m_MOTSDK.ServoTimeCtr (1,5250,1500);
}



void CRDPDlg::OnBTNServoKanan()
{
            // TODO: Add your control notification handler code here
            m_MOTSDK.EnableServo (1);

  m_MOTSDK.ServoTimeCtr (1,1400,1500);
}




void CRDPDlg::SisaPintu()
{
  OnBtnFwd();
  Sleep(2000);
  OnBtnStop();
  //Sleep(3000);
}

void CRDPDlg::MasukRuangan()
{
  OnBtnFwd();
  Sleep(10000);
  OnBtnStop();
}

void CRDPDlg::AmbilGambar()
{
  OnBTNServoAtas();
  OnBTNServoKanan();
            Sleep(2000);
  OnBtnTakePhoto();
}

void CRDPDlg::xxx()
{

            OnMotorSensorEventDrrobotsdkcontrolctrl1();
            belokkirisedikit();
            //OnBtnFwd();
}


void CRDPDlg::yyy()
{

   OnMotorSensorEventDrrobotsdkcontrolctrl1();
   belokkirisedikit();
   //OnBtnFwd();
}


void CRDPDlg::zzz()
{
            AmbilGambar();
            OnBtnStop();
            Sleep(3000);
            SisaPintu();
            OnMotorSensorEventDrrobotsdkcontrolctrl1();
            OnBtnRight();
            MasukRuangan();
}
```

```
void CRDPDlg::CariRuangan()
{
	JlnLurus();
	Sleep(2000);
	OnBtnStop();
}


void CRDPDlg::CariManusia()
{
	if((m_PIR1 > 2000) && (m_PIR2 > 2000))
		belokkirisedikit();

	else
		maju();

}

void CRDPDlg::Base()
{
	//OnBtnRight();
	//OnBtnStop();
	//OnBtnRight();
	//OnBtnFwd();
	//Sleep(20000);
	//if(m_Sonar2 == 255)
	//	OnBtnFwd();
	//if(m_Sonar
}



void CRDPDlg::CariIP()
{


	if(IPclient == "192.168.0.201")
		warna = "red";
	if(IPclient == "192.168.0.202")
```

```
                    warna = "blue";
            if(IPclient == "192.168.0.203")
                    warna = "green";
}
```

SUBPROGRAM

<u>WiRobotSDK</u>

```
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++

// NOTE: Do not modify the contents of this file.  If this class is regenerated by
//  Microsoft Visual C++, your modifications will be overwritten.


#include "stdafx.h"
#include "wirobotsdk.h"

/////////////////////////////////////////////////////////////////////////////
// CWiRobotSDK

IMPLEMENT_DYNCREATE(CWiRobotSDK, CWnd)

/////////////////////////////////////////////////////////////////////////////
// CWiRobotSDK properties

long CWiRobotSDK::GetVoiceSegmentLength()
{
        long result;
        GetProperty(0x1, VT_I4, (void*)&result);
        return result;
}

void CWiRobotSDK::SetVoiceSegmentLength(long propVal)
{
        SetProperty(0x1, VT_I4, propVal);
}

CString CWiRobotSDK::GetRobotName()
```

```
{
	CString result;
	GetProperty(0x2, VT_BSTR, (void*)&result);
	return result;
}

void CWiRobotSDK::SetRobotName(LPCTSTR propVal)
{
	SetProperty(0x2, VT_BSTR, propVal);
}

long CWiRobotSDK::GetTest()
{
	long result;
	GetProperty(0x3, VT_I4, (void*)&result);
	return result;
}

void CWiRobotSDK::SetTest(long propVal)
{
	SetProperty(0x3, VT_I4, propVal);
}

CString CWiRobotSDK::GetTestStr()
{
	CString result;
	GetProperty(0x4, VT_BSTR, (void*)&result);
	return result;
}

void CWiRobotSDK::SetTestStr(LPCTSTR propVal)
{
	SetProperty(0x4, VT_BSTR, propVal);
}

/////////////////////////////////////////////////////////////////////////////
// CWiRobotSDK operations

short CWiRobotSDK::GetSensorSonar1()
{
	short result;
	InvokeHelper(0x5, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorSonar2()
{
	short result;
	InvokeHelper(0x6, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorSonar3()
{
	short result;
	InvokeHelper(0x7, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorSonar4()
{
	short result;
	InvokeHelper(0x8, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorSonar5()
{
	short result;
	InvokeHelper(0x9, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorSonar6()
{
	short result;
	InvokeHelper(0xa, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorHumanMotion1()
{
		short result;
		InvokeHelper(0xb, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorHumanAlarm2()
{
		short result;
		InvokeHelper(0xc, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorHumanMotion2()
{
		short result;
		InvokeHelper(0xd, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorTiltingX()
{
		short result;
		InvokeHelper(0xe, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorTiltingY()
{
		short result;
		InvokeHelper(0xf, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorOverheatAD1()
{
		short result;
		InvokeHelper(0x10, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorOverheatAD2()
```

```
{
        short result;
        InvokeHelper(0x11, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetSensorTemperature()
{
        short result;
        InvokeHelper(0x12, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetSensorIRRange()
{
        short result;
        InvokeHelper(0x13, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetSensorBatteryAD1()
{
        short result;
        InvokeHelper(0x14, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetSensorBatteryAD2()
{
        short result;
        InvokeHelper(0x15, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetSensorRefVoltage()
{
        short result;
        InvokeHelper(0x16, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}
```

```
void CWiRobotSDK::EnableDcMotor(short channel)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x17, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel);
}

void CWiRobotSDK::DisableDcMotor(short channel)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x18, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel);
}

void CWiRobotSDK::EnableServo(short channel)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x19, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel);
}

void CWiRobotSDK::DisableServo(short channel)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x1a, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel);
}

void CWiRobotSDK::SetDcMotorTrajectoryPlan(short channel, short tranPlanMethod)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2;
	InvokeHelper(0x1b, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, tranPlanMethod);
}

void CWiRobotSDK::SetDcMotorSensorFilter(short channel, short filterMethod)
```

```
{
        static BYTE parms[] =
                VTS_I2 VTS_I2;
        InvokeHelper(0x1c, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, filterMethod);
}

void CWiRobotSDK::SetDcMotorSensorUsage(short channel, short sensorType)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2;
        InvokeHelper(0x1d, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, sensorType);
}

void CWiRobotSDK::SetDcMotorControlMode(short channel, short controlMode)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2;
        InvokeHelper(0x1e, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, controlMode);
}

void CWiRobotSDK::DcMotorPositionTimeCtr(short channel, short cmdValue, short timePeriod)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2 VTS_I2;
        InvokeHelper(0x1f, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, cmdValue, timePeriod);
}

void CWiRobotSDK::DcMotorPositionNonTimeCtr(short channel, short cmdValue)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2;
        InvokeHelper(0x20, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, cmdValue);
}

void CWiRobotSDK::DcMotorPwmTimeCtr(short channel, short cmdValue, short timePeriod)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2 VTS_I2;
	InvokeHelper(0x21, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, cmdValue, timePeriod);
}

void CWiRobotSDK::DcMotorPwmNonTimeCtr(short channel, short cmdValue)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2;
	InvokeHelper(0x22, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, cmdValue);
}

void CWiRobotSDK::ServoTimeCtr(short channel, short cmdValue, short timePeriods)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2 VTS_I2;
	InvokeHelper(0x23, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, cmdValue, timePeriods);
}

void CWiRobotSDK::servoNonTimeCtr(short channel, short cmdValue)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2;
	InvokeHelper(0x24, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, cmdValue);
}

short CWiRobotSDK::GetSensorPot(short channel)
{
	short result;
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x25, DISPATCH_METHOD, VT_I2, (void*)&result, parms,
		channel);
	return result;
}

void CWiRobotSDK::PlayAudioFile(LPCTSTR fileName)
```

```
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x26, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 fileName);
}

void CWiRobotSDK::TakePhoto()
{
        InvokeHelper(0x27, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::Refresh()
{
        InvokeHelper(DISPID_REFRESH, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

BOOL CWiRobotSDK::SavePhotoAsBMP(LPCTSTR fileName)
{
        BOOL result;
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x28, DISPATCH_METHOD, VT_BOOL, (void*)&result, parms,
                fileName);
        return result;
}

void CWiRobotSDK::ServoTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4, short cmd5, short
cmd6, short timePeriods)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
        InvokeHelper(0x29, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 cmd1, cmd2, cmd3, cmd4, cmd5, cmd6, timePeriods);
}

short CWiRobotSDK::GetSensorHumanAlarm1()
{
        short result;
        InvokeHelper(0x2a, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetSensorIRCode1()
{
	short result;
	InvokeHelper(0x2b, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorIRCode2()
{
	short result;
	InvokeHelper(0x2c, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorIRCode3()
{
	short result;
	InvokeHelper(0x2d, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorIRCode4()
{
	short result;
	InvokeHelper(0x2e, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

void CWiRobotSDK::DcMotorPositionNonTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4,
short cmd5, short cmd6)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
	InvokeHelper(0x2f, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 cmd1, cmd2, cmd3, cmd4, cmd5, cmd6);
}

void CWiRobotSDK::DcMotorPositionTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4, short
cmd5, short cmd6, short timePeriods)
```

```
{
        static BYTE parms[] =
                VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
        InvokeHelper(0x30, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 cmd1, cmd2, cmd3, cmd4, cmd5, cmd6, timePeriods);
}

void CWiRobotSDK::SetDcMotorPositionControlPID(short channel, short Kp, short Kd, short Ki)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2 VTS_I2 VTS_I2;
        InvokeHelper(0x31, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, Kp, Kd, Ki);
}

void CWiRobotSDK::StartRecord(short voiceSegment)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x32, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 voiceSegment);
}

void CWiRobotSDK::StopRecord()
{
        InvokeHelper(0x33, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

long CWiRobotSDK::GetVoiceSegment()
{
        long result;
        InvokeHelper(0x34, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}

void CWiRobotSDK::ServoNoTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4, short cmd5,
short cmd6)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
        InvokeHelper(0x35, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                        cmd1, cmd2, cmd3, cmd4, cmd5, cmd6);
}

void CWiRobotSDK::DcMotorPwmNonTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4, short
cmd5, short cmd6)
{
            static BYTE parms[] =
                        VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
            InvokeHelper(0x36, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                        cmd1, cmd2, cmd3, cmd4, cmd5, cmd6);
}

void CWiRobotSDK::DcMotorPwmTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4, short
cmd5, short cmd6, short timePeriods)
{
            static BYTE parms[] =
                        VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
            InvokeHelper(0x37, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                        cmd1, cmd2, cmd3, cmd4, cmd5, cmd6, timePeriods);
}

void CWiRobotSDK::DcMotorVelocityNonTimeCtr(short channel, short cmdValue)
{
            static BYTE parms[] =
                        VTS_I2 VTS_I2;
            InvokeHelper(0x38, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                        channel, cmdValue);
}

void CWiRobotSDK::DcMotorVelocityNonTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4,
short cmd5, short cmd6)
{
            static BYTE parms[] =
                        VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
            InvokeHelper(0x39, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                        cmd1, cmd2, cmd3, cmd4, cmd5, cmd6);
}

void CWiRobotSDK::DcMotorVelocityTimeCtrAll(short cmd1, short cmd2, short cmd3, short cmd4, short
cmd5, short cmd6, short timePeriods)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2 VTS_I2;
	InvokeHelper(0x3a, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 cmd1, cmd2, cmd3, cmd4, cmd5, cmd6, timePeriods);
}

void CWiRobotSDK::DcMotorVelocityTimeCtr(short channel, short cmdValue, short timePeriods)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2 VTS_I2;
	InvokeHelper(0x3b, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, cmdValue, timePeriods);
}

void CWiRobotSDK::SetDcMotorVelocityControlPID(short channel, short Kp, short Kd, short Ki)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2 VTS_I2 VTS_I2;
	InvokeHelper(0x3c, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, Kp, Kd, Ki);
}

long CWiRobotSDK::GetImageAddress()
{
	long result;
	InvokeHelper(0x3d, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
	return result;
}

void CWiRobotSDK::StopAudioPlay()
{
	InvokeHelper(0x3e, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

long CWiRobotSDK::GetImageYDataAddress()
{
	long result;
	InvokeHelper(0x3f, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
	return result;
}
```

```
void CWiRobotSDK::LcdDisplayPMS(LPCTSTR bmpFileName)
{
	static BYTE parms[] =
		VTS_BSTR;
	InvokeHelper(0x40, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 bmpFileName);
}

short CWiRobotSDK::GetSensorPotVoltage()
{
	short result;
	InvokeHelper(0x41, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorBatteryAD3()
{
	short result;
	InvokeHelper(0x42, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorPot1()
{
	short result;
	InvokeHelper(0x43, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorPot2()
{
	short result;
	InvokeHelper(0x44, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorPot3()
{
	short result;
	InvokeHelper(0x45, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
```

```
}

short CWiRobotSDK::GetSensorPot4()
{
	short result;
	InvokeHelper(0x46, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorPot5()
{
	short result;
	InvokeHelper(0x47, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetSensorPot6()
{
	short result;
	InvokeHelper(0x48, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

void CWiRobotSDK::SetServoTrajectoryPlan(short channel, short tranPlanMethod)
{
	static BYTE parms[] =
		VTS_I2 VTS_I2;
	InvokeHelper(0x49, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 channel, tranPlanMethod);
}

void CWiRobotSDK::SetCustomDOUT(short dout)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x4a, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 dout);
}

short CWiRobotSDK::GetCustomDIN()
{
	short result;
	InvokeHelper(0x4b, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD1()
{
	short result;
	InvokeHelper(0x4c, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD2()
{
	short result;
	InvokeHelper(0x4d, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD3()
{
	short result;
	InvokeHelper(0x4e, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD4()
{
	short result;
	InvokeHelper(0x4f, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD5()
{
	short result;
	InvokeHelper(0x50, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD6()
```

```
{
	short result;
	InvokeHelper(0x51, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD7()
{
	short result;
	InvokeHelper(0x52, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

short CWiRobotSDK::GetCustomAD8()
{
	short result;
	InvokeHelper(0x53, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
	return result;
}

void CWiRobotSDK::SystemMotorSensorRequest(short Packets)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x54, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 Packets);
}

void CWiRobotSDK::SystemStandardSensorRequest(short Packets)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x55, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 Packets);
}

void CWiRobotSDK::SystemCustomSensorRequest(short Packets)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x56, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                    Packets);
}

void CWiRobotSDK::SetSysMotorSensorPeriod(short PeriodTime)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x57, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 PeriodTime);
}

void CWiRobotSDK::SetSysStandardSensorPeriod(short PeriodTime)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x58, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 PeriodTime);
}

void CWiRobotSDK::SetSysCustomSensorPeriod(short PeriodTime)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x59, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 PeriodTime);
}

void CWiRobotSDK::EnableMotorSensorSending()
{
        InvokeHelper(0x5a, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::EnableStandardSensorSending()
{
        InvokeHelper(0x5b, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::EnableCustomSensorSending()
{
        InvokeHelper(0x5c, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}
```

```
void CWiRobotSDK::DisableMotorSensorSending()
{
	InvokeHelper(0x5d, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::DisableStandardSensorSending()
{
	InvokeHelper(0x5e, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::DisableCustomSensorSending()
{
	InvokeHelper(0x5f, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::SetSysAllSensorPeriod(short PeriodTime)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x60, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 PeriodTime);
}

void CWiRobotSDK::SystemAllSensorRequest(short Packets)
{
	static BYTE parms[] =
		VTS_I2;
	InvokeHelper(0x61, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
		 Packets);
}

void CWiRobotSDK::EnableAllSensorSending()
{
	InvokeHelper(0x62, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::DisableAllSensorSending()
{
	InvokeHelper(0x63, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}
```

```
long CWiRobotSDK::GetVoiceSegLength()
{
        long result;
        InvokeHelper(0x64, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}

void CWiRobotSDK::SetInfraredControlOutput(short low, short high)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2;
        InvokeHelper(0x65, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 low, high);
}

short CWiRobotSDK::GetMotorCurrent1()
{
        short result;
        InvokeHelper(0x66, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetMotorCurrent2()
{
        short result;
        InvokeHelper(0x67, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetMotorCurrent3()
{
        short result;
        InvokeHelper(0x68, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetMotorCurrent4()
{
        short result;
        InvokeHelper(0x69, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
```

```
		return result;
}

short CWiRobotSDK::GetMotorCurrent5()
{
		short result;
		InvokeHelper(0x6a, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetMotorCurrent6()
{
		short result;
		InvokeHelper(0x6b, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetEncoderPulse1()
{
		short result;
		InvokeHelper(0x6c, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetEncoderSpeed1()
{
		short result;
		InvokeHelper(0x6d, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetEncoderPulse2()
{
		short result;
		InvokeHelper(0x6e, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetEncoderSpeed2()
{
		short result;
		InvokeHelper(0x6f, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetEncoderDir1()
{
		short result;
		InvokeHelper(0x70, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetEncoderDir2()
{
		short result;
		InvokeHelper(0x71, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetSensorSonar(short channel)
{
		short result;
		static BYTE parms[] =
				VTS_I2;
		InvokeHelper(0x72, DISPATCH_METHOD, VT_I2, (void*)&result, parms,
				channel);
		return result;
}

short CWiRobotSDK::GetMotorCurrent(short channel)
{
		short result;
		static BYTE parms[] =
				VTS_I2;
		InvokeHelper(0x73, DISPATCH_METHOD, VT_I2, (void*)&result, parms,
				channel);
		return result;
}

short CWiRobotSDK::GetCustomAD(short channel)
{
		short result;
```

```
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x74, DISPATCH_METHOD, VT_I2, (void*)&result, parms,
                channel);
        return result;
}

void CWiRobotSDK::SetMotorPolarity1(short Polarity)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x75, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 Polarity);
}

void CWiRobotSDK::SetMotorPolarity2(short Polarity)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x76, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 Polarity);
}

void CWiRobotSDK::SetMotorPolarity3(short Polarity)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x77, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 Polarity);
}

void CWiRobotSDK::SetMotorPolarity4(short Polarity)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x78, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 Polarity);
}

void CWiRobotSDK::SetMotorPolarity5(short Polarity)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x79, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 Polarity);
}

void CWiRobotSDK::SetMotorPolarity6(short Polarity)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x7a, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 Polarity);
}

void CWiRobotSDK::SetMotorPolarity(short channel, short Polarity)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2;
        InvokeHelper(0x7b, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel, Polarity);
}

void CWiRobotSDK::EnableBumperProtection()
{
        InvokeHelper(0x7c, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::DisableBumperProtection()
{
        InvokeHelper(0x7d, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CWiRobotSDK::LcdDisplayPMB(LPCTSTR bmpFileName)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x7e, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 bmpFileName);
}

void CWiRobotSDK::SuspendDcMotor(short channel)
```

```
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x7f, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel);
}

void CWiRobotSDK::ResumeDcMotor(short channel)
{
        static BYTE parms[] =
                VTS_I2;
        InvokeHelper(0x80, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 channel);
}

void CWiRobotSDK::connectRobot(LPCTSTR robotName)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x81, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 robotName);
}

long CWiRobotSDK::GetVoiceData(long VoicePtr)
{
        long result;
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x82, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                VoicePtr);
        return result;
}

short CWiRobotSDK::getSenID1()
{
        short result;
        InvokeHelper(0x83, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::getSenID2()
```

```
{
		short result;
		InvokeHelper(0x84, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::getSenID3()
{
		short result;
		InvokeHelper(0x85, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::getSenID4()
{
		short result;
		InvokeHelper(0x86, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetGPS01()
{
		short result;
		InvokeHelper(0x87, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetGPS02()
{
		short result;
		InvokeHelper(0x88, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}

short CWiRobotSDK::GetGPS03()
{
		short result;
		InvokeHelper(0x89, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
		return result;
}
```

```
short CWiRobotSDK::GetGPS04()
{
        short result;
        InvokeHelper(0x8a, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetGPS05()
{
        short result;
        InvokeHelper(0x8b, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetGPS06()
{
        short result;
        InvokeHelper(0x8c, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetGPS07()
{
        short result;
        InvokeHelper(0x8d, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

short CWiRobotSDK::GetGPS08()
{
        short result;
        InvokeHelper(0x8e, DISPATCH_METHOD, VT_I2, (void*)&result, NULL);
        return result;
}

void CWiRobotSDK::SetGPSID(short ID1, short ID2, short ID3, short ID4)
{
        static BYTE parms[] =
                VTS_I2 VTS_I2 VTS_I2 VTS_I2;
        InvokeHelper(0x8f, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 ID1, ID2, ID3, ID4);
}
```

SUBPROGRAM

SOCKET

```
#include "stdafx.h"
#include "Socket.h"

/*
---------------------------------------------------------------------------------
Using these conventions results in better readable code and less coding errors !
---------------------------------------------------------------------------------

     cName  for generic class definitions
     CName  for MFC     class definitions
     tName  for type    definitions
     eName  for enum    definitions
     kName  for struct  definitions

    e_Name  for enum variables
    E_Name  for enum constant values

    i_Name  for instances of classes
    h_Name  for handles

    T_Name  for Templates
    t_Name  for TCHAR or LPTSTR

    s_Name  for strings
   sa_Name  for Ascii strings
   sw_Name  for Wide (Unicode) strings
   bs_Name  for BSTR
    f_Name  for function pointers
    k_Name  for contructs (struct)

    b_Name  bool,BOOL 1 Bit
```

```
  s8_Name    signed  8 Bit (char)
 s16_Name    signed 16 Bit (SHORT, WCHAR)
 s32_Name    signed 32 Bit (LONG, int)
 s64_Name    signed 64 Bit (LONGLONG)

  u8_Name  unsigned  8 Bit (BYTE)
 u16_Name  unsigned 16 bit (WORD)
 u32_Name  unsigned 32 Bit (DWORD, UINT)
 u64_Name  unsigned 64 Bit (ULONGLONG)

   d_Name  for double

 ----------------

   m_Name  for member variables of a class (e.g. ms32_Name for int member variable)
   g_Name  for global (static) variables   (e.g. gu16_Name for global WORD)
   p_Name  for pointer                     (e.g.   ps_Name  *pointer to string)
  pp_Name  for pointer to pointer          (e.g.  ppd_Name **pointer to double)
*/

TCP::cSocket::cSocket()
{
	mb_Initialized = FALSE;
	mu32_WaitIndex = 0;
}

TCP::cSocket::~cSocket()
{
	if (mb_Initialized)
	{
		Close();
		WSACleanup();
	}
}

// protected
// Load ws2_32.dll and initialize Windsock 2.0
DWORD TCP::cSocket::Initialize()
{
	if (mb_Initialized)
```

```
		return 0;

	// Winsock version 2.0 is available on ALL Windows operating systems
	// except Windows 95 which comes with Winsock 1.1
	WSADATA k_Data;
	DWORD u32_Err = WSAStartup(MAKEWORD(2,0), &k_Data);

	mb_Initialized = (u32_Err == 0);
	return u32_Err;
}

// Closes all open sockets
DWORD TCP::cSocket::Close()
{
	if (!mi_List.mu32_Count)
		return WSAENOTCONN; // no socket open

	// Request thread safe access to manipulate mi_List
	cLock i_Lock;
	DWORD u32_Err = i_Lock.Request(&mk_Lock);
	if (u32_Err)
		return u32_Err;

	mi_List.CloseAll();
	return 0;
}

// returns the current state of the socket
TCP::cSocket::eState TCP::cSocket::GetState()
{
	return mi_List.me_State;
}

// Get the count of open sockets
DWORD TCP::cSocket::GetSocketCount()
{
	return mi_List.mu32_Count;
}

// retrieve the socket and it's peer IP at the given index from the socket list
// pu32_IP = 0x6401a8c0 -> 192.168.1.100
```

```
DWORD TCP::cSocket::GetSocket(DWORD u32_Index, SOCKET* ph_Socket, DWORD* pu32_IP)
{
        if (u32_Index >= mi_List.mu32_Count)
                return ERROR_INVALID_PARAMETER;

        if (ph_Socket) *ph_Socket = mi_List.mk_Data[u32_Index].h_Socket;
        if (pu32_IP)   *pu32_IP   = mi_List.mk_Data[u32_Index].u32_IP;
        return 0;
}

// protected
// Create a new unbound socket and add it to mi_List at index 0
DWORD TCP::cSocket::CreateSocket()
{
        DWORD u32_Err = Initialize();
        if (u32_Err)
                return u32_Err;

        if (mi_List.mu32_Count)
                return WSAEISCONN; // Socket already created

        SOCKET h_Socket = socket(AF_INET, SOCK_STREAM, 0);
        if (h_Socket == INVALID_SOCKET)
                return WSAGetLastError();

        HANDLE h_Event = WSACreateEvent();
        if (h_Event == WSA_INVALID_EVENT)
        {
                DWORD u32_Err = WSAGetLastError();
                closesocket(h_Socket);
                return u32_Err;
        }

        // Monitor all events on the socket
        if (WSAEventSelect(h_Socket, h_Event, FD_ALL_EVENTS) == SOCKET_ERROR)
        {
                u32_Err = WSAGetLastError();
                closesocket  (h_Socket);
                WSACloseEvent(h_Event);
                return u32_Err;
        }
```

```
		mi_List.Add(h_Socket, h_Event);
		return 0;
}

// Creates a Server socket
// You must wait for FD_ACCEPT events before sending data
// u32_BindIP = 0          --> listen on all network adapters
// u32_BindIP = 10.1.0.143 --> listen only on the network adapter with local IP 10.1.0.143
// u32_BindIP = 10.1.2.208 --> listen only on the network adapter with local IP 10.1.2.208
DWORD TCP::cSocket::Listen(DWORD u32_BindIP, USHORT u16_Port)
{
		// Create a server socket which waits for Accept events
		// This socket itself will never be connected to any client!
		DWORD u32_Err = CreateSocket();
		if (u32_Err)
				return u32_Err;

		// get the new socket's data structure
		kData* pk_Data = &mi_List.mk_Data[0];

		SOCKADDR_IN k_Addr;
		k_Addr.sin_family      = AF_INET;
		k_Addr.sin_addr.s_addr = u32_BindIP;
		k_Addr.sin_port        = htons(u16_Port);

		// Bind the socket to the given port
		if     (bind(pk_Data->h_Socket,     (LPSOCKADDR)&k_Addr,     sizeof(SOCKADDR_IN))     ==
SOCKET_ERROR)
		{
				DWORD u32_Err = WSAGetLastError();
				mi_List.CloseAll();
				return u32_Err;
		}

		// Start listening for connection requests
		if (listen(pk_Data->h_Socket, WSA_MAXIMUM_WAIT_EVENTS) == SOCKET_ERROR)
		{
				DWORD u32_Err = WSAGetLastError();
				mi_List.CloseAll();
				return u32_Err;
```

```
        }

        // The server is not yet connected (wait for FD_ACCEPT!)
        mi_List.me_State = E_Server;
        return 0;
}

// Creates a Client socket
// u32_ServIP = 0x6401a8c0 -> 192.168.1.100
// *************************** ATTENTION ************************************
// When this funcion returns without error the socket is NOT YET connected!
// You must wait for the FD_CONNECT event before sending data to this socket!
// *************************** ATTENTION ************************************
DWORD TCP::cSocket::ConnectTo(DWORD u32_ServIP, USHORT u16_Port)
{
        // Create a client socket which will connect to the server
        DWORD u32_Err = CreateSocket();
        if (u32_Err)
                return u32_Err;

        // get the new socket's data structure
        kData* pk_Data = &mi_List.mk_Data[0];

        SOCKADDR_IN k_Addr;
        k_Addr.sin_family      = AF_INET;
        k_Addr.sin_addr.s_addr = u32_ServIP;
        k_Addr.sin_port        = htons(u16_Port);

        // Connect the socket to the given IP and port
        if (connect(pk_Data->h_Socket, (LPSOCKADDR)&k_Addr, sizeof(SOCKADDR_IN)) ==
SOCKET_ERROR)
        {
                DWORD u32_Err = WSAGetLastError();
                if (u32_Err != WSAEWOULDBLOCK)
                {
                        mi_List.CloseAll();
                        return u32_Err;
                }
        }

        // The client is not yet connected (wait for FD_CONNECT!)
        mi_List.me_State = E_Client;

        pk_Data->u32_IP = u32_ServIP;
        return 0;
}

// Waits for incoming events on the port and processes them (used on Server + Client)
// returns the event(s) that occurred and the socket and it's IP-address which has caused the event.
// If the event is FD_READ the data will be read into s8_RecvBuf and pu32_Read will receive the count of
bytes read.
// If the event is FD_WRITE the remaining data in the send buffer will be sent and pu32_Sent receives the
bytes sent.
// If there is more data to be read or sent, the next call to ProcessEvents() will process the next block of data.
// returns ERROR_TIMEOUT if during the given timeout no event occurres
// pu32_IP = 0x6401a8c0 -> 192.168.1.100
DWORD TCP::cSocket::ProcessEvents(DWORD   u32_Timeout, // IN
                 DWORD* pu32_Event,   // OUT
                 DWORD* pu32_IP,      // OUT
                                                            SOCKET*    ph_Socket,    //
OUT
                 char*    s8_RecvBuf, // OUT
                 DWORD   u32_BufLen,  // IN
                 DWORD* pu32_Read,    // OUT
                                                            DWORD*  pu32_Sent)       //
OUT
{
        *ph_Socket  = 0;
        *pu32_Event = 0;
        *pu32_IP    = 0;
        *pu32_Read  = 0;
        *pu32_Sent  = 0;

        // Block here if SendTo() or Close() have requested thread safe access to manipulate mi_List.
        cLock i_Lock;
        DWORD u32_Err = i_Lock.Loop(&mk_Lock);
        if (u32_Err)
                return u32_Err;

        if (!mi_List.mu32_Count)
                return WSAENOTCONN; // No socket open or just closed while Loop() was blocking
        // The first event is used to escape from WaitForMultiplEvents in cLock::Request()
        mi_List.mh_Events[0] = mk_Lock.h_ExitEvent;

        // Wait until an event occurred or the timeout has elapsed
        // u32_Index is the index in the socket list of the event that has occurred
        DWORD      u32_Index      =      WSAWaitForMultipleEventsEx(mi_List.mu32_Count+1,
&mu32_WaitIndex, mi_List.mh_Events, u32_Timeout);

        if (u32_Index == WSA_WAIT_FAILED)
                return WSAGetLastError();

        if (u32_Index == WSA_WAIT_TIMEOUT)
                return ERROR_TIMEOUT;

        u32_Index -= WSA_WAIT_EVENT_0;

        // mi_List.mh_Events[0] is used for the lock. It is not associated with a socket.
        if (u32_Index == 0)
                return 0;

        // Convert the 1-based event index into the zero-based kData index
        u32_Index--;

        // Get the data associated with the socket that has signaled the event
        kData* pk_Data = &mi_List.mk_Data[u32_Index];

        // Get the event(s) that occurred and their associated error array
        WSANETWORKEVENTS k_Events;
        if (WSAEnumNetworkEvents(pk_Data->h_Socket, mi_List.mh_Events[u32_Index+1], &k_Events)
== SOCKET_ERROR)
        {
                DWORD u32_Err = WSAGetLastError();
                mi_List.Remove(u32_Index); // remove socket with problem
                return u32_Err;
        }

        *pu32_Event = k_Events.lNetworkEvents;

        // ------------------------------------------

        if (k_Events.lNetworkEvents & FD_ACCEPT)
```

```
	{
		if (k_Events.iErrorCode[FD_ACCEPT_BIT])
			return k_Events.iErrorCode[FD_ACCEPT_BIT];

		SOCKADDR_IN k_Addr;
		int s32_Len = sizeof(k_Addr);
		// Accept the connect request from a client (k_Addr receives peer IP of connecting client)
		// The callback AcceptCondition() checks if the maximum count of connected clients was
exceeded
		// If there are already 63 sockets open, the connect request is rejected.
		SOCKET  h_Socket  =  WSAAccept(pk_Data->h_Socket,  (LPSOCKADDR)&k_Addr,
&s32_Len, AcceptCondition, (DWORD_PTR)this);
		if (h_Socket == INVALID_SOCKET)
			return WSAGetLastError();

		HANDLE h_Event = WSACreateEvent();
		if (h_Event == WSA_INVALID_EVENT)
		{
			DWORD u32_Err = WSAGetLastError();
			closesocket(h_Socket);
			return u32_Err;
		}

		// Monitor events on the newly connected client socket
		if (WSAEventSelect(h_Socket, h_Event, FD_ALL_EVENTS) == SOCKET_ERROR)
		{
			DWORD u32_Err = WSAGetLastError();
			closesocket  (h_Socket);
			WSACloseEvent(h_Event);
			return u32_Err;
		}

		// Append the new socket to the socket list
		pk_Data = mi_List.Add(h_Socket, h_Event);
		// Store the client's IP in the socket list
		pk_Data->u32_IP = k_Addr.sin_addr.s_addr;

		// successfully connected
		mi_List.me_State = (eState)(E_Server | E_Connected);

		// Do not return 0 here (multiple events may be set!!)
```

```
        }

        *ph_Socket = pk_Data->h_Socket;
        *pu32_IP   = pk_Data->u32_IP;

        // -------------------------------------------

        if (k_Events.lNetworkEvents & FD_CONNECT)
        {
                if (k_Events.iErrorCode[FD_CONNECT_BIT])
                {
                        // The connection has failed -> remove faulty socket from the socket list
                        mi_List.Remove(u32_Index);
                        return k_Events.iErrorCode[FD_CONNECT_BIT];
                }

                // successfully connected
                mi_List.me_State = (eState)(E_Client | E_Connected);

                // Do not return 0 here (multiple events may be set!!)
        }

        // -------------------------------------------

        if (k_Events.lNetworkEvents & FD_READ)
        {
                if (k_Events.iErrorCode[FD_READ_BIT])
                {
                        mi_List.Remove(u32_Index); // remove socket with Read error
                        return k_Events.iErrorCode[FD_READ_BIT];
                }

                // Read the data into the read buffer
                DWORD u32_Flags = 0;
                WSABUF  k_Buf;
                k_Buf.buf = s8_RecvBuf;
                k_Buf.len = u32_BufLen;
                if  (WSARecv(pk_Data->h_Socket,  &k_Buf,  1,  pu32_Read,  &u32_Flags,  0,  0)  ==
SOCKET_ERROR)
                {
                        DWORD u32_Err = WSAGetLastError();
                        if (u32_Err && u32_Err != WSAEWOULDBLOCK)
                        {
                                mi_List.Remove(u32_Index); // remove socket with Read error
                                return u32_Err;
                        }
                }

                // Do not return 0 here (multiple events may be set!!)
        }

        // -------------------------------------------

        if (k_Events.lNetworkEvents & FD_WRITE)
        {
                if (k_Events.iErrorCode[FD_WRITE_BIT])
                {
                        mi_List.Remove(u32_Index); // remove socket with Write error
                        return k_Events.iErrorCode[FD_WRITE_BIT];
                }

                // Is there pending data in the send buffer waiting to be sent ?
                if (pk_Data->s8_SendBuf)
                {
                        // Send as much as possible data from the send buffer
                        DWORD u32_Before = pk_Data->u32_SendPos;
                        DWORD   u32_Err   =   SendDataBlock(pk_Data->h_Socket,   pk_Data-
>s8_SendBuf, &pk_Data->u32_SendPos, pk_Data->u32_SendLen);
                        *pu32_Sent = pk_Data->u32_SendPos - u32_Before;

                        if (pk_Data->u32_SendPos == pk_Data->u32_SendLen)
                        {
                                // All data has been sent successfully -> delete the buffer
                                delete pk_Data->s8_SendBuf;
                                pk_Data->s8_SendBuf = 0;
                        }

                        if (u32_Err && u32_Err != WSAEWOULDBLOCK)
                        {
                                mi_List.Remove(u32_Index); // remove socket with Write error
                                return u32_Err;
                }
            }
            // Do not return 0 here (multiple events may be set!!)
        }

        // -------------------------------------------

        if (k_Events.lNetworkEvents & FD_CLOSE)
        {
            // The socket has closed (gracefully or with error) -> remove it from the socket list
            mi_List.Remove(u32_Index);

            if (k_Events.iErrorCode[FD_CLOSE_BIT]) // e.g. WSAECONNABORTED
                return k_Events.iErrorCode[FD_CLOSE_BIT];

            // Do not return 0 here (multiple events may be set!!)
        }

        // -------------------------------------------

        return 0;
}

// protected:
// This is a special Wait function which eliminates a problem of WSAWaitForMultipleEvents:
// WSAWaitForMultipleEvents scans the events in the event array ph_Events from zero on.
// When it finds one that is signaled it stops and retruns it's index.
// This may cause a problem on a server with high load near 100% CPU power:
// When multiple events are signaled always the ones at the begin of the event array will be preferred
// and the events at the end of the array will be in disadvantage.
// To avoid this, this function uses a pointer pu32_Index from which on a signaled event is searched.
// This pointer is incremented until the last event and then starts from the first again.
// So every client on a server has the same priority in being served.
// ph_Events  = array of event handles
// u32_Count  = count of events in array
// pu32_Index = rotating index
DWORD TCP::cSocket::WSAWaitForMultipleEventsEx(DWORD u32_Count, DWORD* pu32_Index,
WSAEVENT* ph_Events, DWORD u32_Timeout)
{
        // Here *pu32_Index is at the position where the last time an event has been signaled
        // Search for a signaled event from *pu32_Index +1 upwards
```

```
	for (DWORD C=0; C<u32_Count; C++)
	{
		(*pu32_Index)++;

		if (*pu32_Index >= u32_Count)
			*pu32_Index = 0;

		// Check if the event is set (Timeout = 0)
		DWORD u32_Res = WaitForSingleObject(ph_Events[*pu32_Index], 0);
		if (u32_Res == WAIT_OBJECT_0)
			return WSA_WAIT_EVENT_0 + *pu32_Index;
	}

	// There is no event signaled -> this means that the server is not under stress
	// There is no reason to check the events one by one anymore.
	DWORD u32_Res = WSAWaitForMultipleEvents(u32_Count, ph_Events, FALSE, u32_Timeout,
FALSE);
	if (u32_Res != WSA_WAIT_FAILED && u32_Res != WSA_WAIT_TIMEOUT)
	{
		*pu32_Index = u32_Res - WSA_WAIT_EVENT_0;
	}
	return u32_Res;
}

// static
// Decides if a connection request from a client is accepted. (Max 63 open sockets possible)
// WSAAccept() will return WSAECONNREFUSED if AcceptCondition() returns CF_REJECT
int WINAPI TCP::cSocket::AcceptCondition(WSABUF* pk_CallerId, WSABUF* pk_CallerData, QOS*
pk_SQOS, QOS* pk_GQOS,

WSABUF* pk_CalleeId, WSABUF* pk_CalleeData, UINT* pu32_Group, DWORD_PTR p_Param)
{
	cSocket* p_This = (cSocket*)p_Param;

	if (p_This->mi_List.mu32_Count >= WSA_MAXIMUM_WAIT_EVENTS-1)
		return CF_REJECT;
	else
		return CF_ACCEPT;
}

// protected
// Send the remaining data and adjust the pu32_Pos pointer.
// If the data was sent only partially (WSAEWOULDBLOCK) this function must be called again
DWORD TCP::cSocket::SendDataBlock(SOCKET h_Socket, char* s8_Buf, DWORD* pu32_Pos, DWORD
u32_Len)
{
        while (*pu32_Pos < u32_Len)
        {
                WSABUF k_Buf;
                k_Buf.buf =  s8_Buf + *pu32_Pos;
                k_Buf.len = u32_Len - *pu32_Pos;

                DWORD u32_Sent = 0;
                if (WSASend(h_Socket, &k_Buf, 1, &u32_Sent, 0, 0, 0) == SOCKET_ERROR)
                        return WSAGetLastError();

                *pu32_Pos += u32_Sent;
        };
        return 0;
}

// Send the data in s8_Buf to the given socket (Server + Client)
// This function will not block.
// If the data cannot be sent immediately, the function returns WSAEWOULDBLOCK and the data is
buffered until the next
// FD_WRITE event which will be signaled when the correct time has come to send more data to that socket.
// It is possible that a part of the data is sent immediately and the rest is buffered to be sent after next
FD_WRITE.
// If you call this function while a previous Send operation is still pending, the function returns
WSA_IO_PENDING.
// In this case try again later!
// ATTENTION:
// When this function returns without error, this does not mean that all data has already arrived at the
recipient!
// WinSock uses a transport buffer and the real transmision of the data may take a long time after this function
has returned.
DWORD TCP::cSocket::SendTo(SOCKET h_Socket, char* s8_Buf, DWORD u32_Len)
{
        // Request thread safe access to manipulate mi_List.
        cLock i_Lock;
        DWORD u32_Err = i_Lock.Request(&mk_Lock);
        if (u32_Err)
```

```
                return u32_Err;

        if (!(mi_List.me_State & E_Connected))
                return WSAENOTCONN; // Socket is not connected

        int s32_Index = mi_List.FindSocket(h_Socket);
        if (s32_Index < 0)
                return ERROR_INVALID_PARAMETER; // Invalid Socket handle passed

        kData* pk_Data = &mi_List.mk_Data[s32_Index];

        if (pk_Data->s8_SendBuf)
                return WSA_IO_PENDING; // a Send operation is still in progress on this socket

        // Sends as much data as possible at this moment, increases u32_Pos
        DWORD u32_Pos = 0;
        u32_Err = SendDataBlock(h_Socket, s8_Buf, &u32_Pos, u32_Len);

        // Not all the data could be sent right now:
        // The remaining data is copied into a buffer and will be sent when FD_WRITE becomes signaled.
        if (u32_Err == WSAEWOULDBLOCK)
        {
                u32_Len -= u32_Pos;
                s8_Buf  += u32_Pos;

                pk_Data-> s8_SendBuf = new char[u32_Len];
                pk_Data->u32_SendLen = u32_Len;
                pk_Data->u32_SendPos = 0;
                memcpy(pk_Data->s8_SendBuf, s8_Buf, u32_Len);
                return u32_Err;
        }

        if (u32_Err && u32_Err != WSA_IO_PENDING)
                mi_List.Remove(s32_Index); // remove socket with Write error

        return u32_Err;
}

// LocalIp = 0x6401a8c0 -> 192.168.1.100
// returns a list of all local IP's on this computer (multiple IP's if multiple network adapters)
DWORD TCP::cSocket::GetLocalIPs(cArray<DWORD>* pi_IpList)
```

```
{
        DWORD u32_Err = Initialize();
        if (u32_Err)
                return u32_Err;

        char s8_Host[500];
        if (gethostname(s8_Host, sizeof(s8_Host)) == SOCKET_ERROR)
                return WSAGetLastError();

        struct hostent* pk_Host = gethostbyname(s8_Host);
        if (!pk_Host)
                return WSAGetLastError();

        // The IP list is zero terminated
        DWORD u32_Count = 0;
        while (pk_Host->h_addr_list[u32_Count]) u32_Count++;

        if (!u32_Count)
                return WSAENETDOWN; // no local IP means no network available

        pi_IpList->Allocate(u32_Count);

        for (DWORD i=0; i<u32_Count; i++)
        {
                pi_IpList->Set(i, *((DWORD*)pk_Host->h_addr_list[i]));
        }
        return 0;
}

/*************************************************************************************

        embedded class cList

        Description:
        Stores for all open sockets: their Handle, Wait Event, IP address, Write buffer for pending data

        CLIENT:
        - uses only Index=0 which holds the socket that is connect to the server.

        SERVER:
```

- uses a socket at Index=0 which only waits for incomming connection requests. (always: mu32_IP[0]==0)

- The socket at Index=0 is never connected to any client.

- Each of the following sockets Index=1,2,3... may be connected to one client.

Author:

Elmü (www.netcult.ch/elmue)

```
****************************************************************************************
/

TCP::cSocket::cList::cList()
{
	mu32_Count = 0;
	me_State   = E_Disconnected;
}

TCP::cSocket::cList::~cList()
{
	CloseAll();
}

void TCP::cSocket::cList::CloseAll()
{
	while (mu32_Count)
	{
		Remove(mu32_Count-1);
	}
}

TCP::cSocket::kData* TCP::cSocket::cList::Add(SOCKET h_Socket, HANDLE h_Event)
{
	// Store max 63 sockets
	if (mu32_Count >= WSA_MAXIMUM_WAIT_EVENTS-1)
		return 0;

	memset(&mk_Data[mu32_Count], 0, sizeof(kData));

	mk_Data  [mu32_Count].h_Socket = h_Socket;
	// mh_Events[0] is used for the lock. It is not associated with a socket.
	mh_Events[mu32_Count+1] = h_Event;
```

```
        return &mk_Data[mu32_Count++];
}

BOOL TCP::cSocket::cList::Remove(DWORD u32_Index)
{
        if (u32_Index >= mu32_Count)
                return FALSE;

        shutdown   (mk_Data[u32_Index].h_Socket, SD_BOTH);
        closesocket(mk_Data[u32_Index].h_Socket);

        // mh_Events[0] is used for the lock. It is not associated with a socket.
        WSACloseEvent(mh_Events[u32_Index+1]);

        if (mk_Data[u32_Index].s8_SendBuf)
                delete mk_Data[u32_Index].s8_SendBuf;

        // Close the gap by shifting down
        for (DWORD i=u32_Index; i<mu32_Count-1; i++)
        {
                mk_Data  [i]   = mk_Data  [i+1];
                mh_Events[i+1] = mh_Events[i+2];
        }

        mu32_Count--;

        if (mu32_Count==0)
                me_State = E_Disconnected;

        // remove E_Connected flag from the server (Socket[0] is never connected)
        if (mu32_Count==1 && (me_State & E_Server))
                me_State = E_Server;

        return TRUE;
}

// returns the index of the given socket in the socket list
// returns -1 if not found
int TCP::cSocket::cList::FindSocket(SOCKET h_Socket)
{
	for (DWORD i=0; i<mu32_Count; i++)
	{
		if (mk_Data[i].h_Socket == h_Socket)
			return i;
	}
	return -1;
}

/************************************************************************************

	embedded struct kLock and class cLock

	Description:
	This class is used to mutually lock the access to mi_List if cSocket is running multithreaded.
	cLock must be created on the stack in a function requesting write access to mi_List:
	The function Loop()    locks when entering into the endless loop ProcessEvents()
	The function Request() locks when entering into Close() or SendTo(),
	The destructor of cLock releases the lock when these functions have exited.

	ATTENTION:
	A Mutex alone will not work here because ProcessEvents() is an endless loop,
	which re-enters before the thread context has been switched to the other thread!

	If multithreading is not in use this class does not block in neither function.

	Author:
	Elmü (www.netcult.ch/elmue)

*************************************************************************************
/

TCP::cSocket::kLock::kLock()
{
	h_LoopEvent = 0;
	h_ExitEvent = 0;
	h_Mutex     = 0;
}

TCP::cSocket::kLock::~kLock()
{
	CloseHandle(h_LoopEvent);
```

```
        CloseHandle(h_ExitEvent);
        CloseHandle(h_Mutex);
}

DWORD TCP::cSocket::kLock::Init()
{
        if (!h_LoopEvent) h_LoopEvent = CreateEvent(0, TRUE,  TRUE,  0); // manual-reset, default= run
        if (!h_LoopEvent) return GetLastError();

        if (!h_ExitEvent) h_ExitEvent = CreateEvent(0, FALSE, FALSE, 0); // auto-reset,   default= block
        if (!h_ExitEvent) return GetLastError();

        if (!h_Mutex)     h_Mutex     = CreateMutex(0, FALSE, 0);
        if (!h_Mutex)     return GetLastError();

        return 0;
}

// -------------------------------------------------

TCP::cSocket::cLock::cLock()
{
        mh_Mutex = 0;
}

// Blocks a Request in SendTo() and Close()
// If cSocket is used single-threaded, Request() will never block
DWORD TCP::cSocket::cLock::Request(kLock* pk_Lock)
{
        DWORD u32_Err = pk_Lock->Init();
        if (u32_Err)
                return u32_Err;

        // FIRST: Block ProcessEvents() the next time BEFORE WaitForMultipleEvents() is reached
        if (!ResetEvent(pk_Lock->h_LoopEvent))
                return GetLastError();

        // SECOND: Escape from a blocking WaitForMultipleEvents()
        if (!SetEvent(pk_Lock->h_ExitEvent))
                return GetLastError();
        // THIRD: Wait until ProcessEvents() has exited (only if multithreading)
        if (WaitForSingleObject(pk_Lock->h_Mutex, INFINITE) == WAIT_FAILED)
                return GetLastError();

        // Now we grabbed the Mutex -> allow ProcessEvents to continue
        if (!SetEvent(pk_Lock->h_LoopEvent))
                return GetLastError();

        // Srore the mutex to be released in the destructor
        mh_Mutex = pk_Lock->h_Mutex;
        return 0;
}

// Blocks the endless loop ProcessEvents() after a Request() was made.
// If cSocket is used single-threaded, Loop() will never block
DWORD TCP::cSocket::cLock::Loop(kLock* pk_Lock)
{
        DWORD u32_Err = pk_Lock->Init();
        if (u32_Err)
                return u32_Err;

        // The following line normally does not block (Event always set). Only after a Request() it
        // will block to give Request() a chance to grab the Mutex before the endless loop re-enters
        if (WaitForSingleObject(pk_Lock->h_LoopEvent, INFINITE) == WAIT_FAILED)
                return GetLastError();

        // Wait until SendTo() or Close() have exited
        if (WaitForSingleObject(pk_Lock->h_Mutex, INFINITE) == WAIT_FAILED)
                return GetLastError();

        mh_Mutex = pk_Lock->h_Mutex;
        return 0;
}

// Destructor is executed, when the calling function (SendTo(), Close(), ProcessEvents()) has exited
TCP::cSocket::cLock::~cLock()
{
        // Allow the other thread to continue its work
        ReleaseMutex(mh_Mutex);
}
```

SUBPROGRAM

DIB

```
// Dib.cpp: implementation of the CDib class.
//
//////////////////////////////////////////////////////////////////////

#include "Dib.h"

IMPLEMENT_DYNAMIC(CDib, CBitmap)

CDib::CDib()
{
m_pbmih = NULL;
m_hdd = NULL;
}

CDib::~CDib()
{
DeleteObject();
}

void CDib::DeleteObject()

{
if (m_hdd)
{
DrawDibClose(m_hdd);
m_hdd = NULL;
}

if (m_pbmih)
{
delete [] (char*)m_pbmih;
m_pbmih = NULL;
}
}
```

```
UINT CDib::GetNumPaletteColors()
{
// Calculate # entries in color table:
// if biClrUsed is not specified, then use:
// (2,16,256) for (1,4,8)-bit bitmaps;
// 0 for 24, 32-bit bitmaps (no color table)

UINT nColors=m_pbmih->biClrUsed;
if (nColors==0 && m_pbmih->biBitCount<=8)
nColors = 1<<m_pbmih->biBitCount;

return nColors;
}

LPBYTE CDib::GetBits()
{
return (LPBYTE)m_pbmih + m_pbmih->biSize + GetNumPaletteColors()*sizeof(RGBQUAD);
}

const BITMAP_TYPE = (WORD)('M' << 8) | 'B';

BOOL CDib::Load(LPCTSTR szPathName)
{
if( m_hObject )
Detach();

if( Attach(::LoadImage(NULL, szPathName,IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE |
LR_CREATEDIBSECTION | LR_DEFAULTSIZE)) )
{
CFile file;
BOOL bRet = FALSE;

file.Open( szPathName, CFile::modeRead );
if( Load( file )==TRUE )
bRet = CreatePalette();
file.Close();
return bRet;
}

return FALSE;
}
```

```
BOOL CDib::Load(CFile &file)
{
BITMAPFILEHEADER hdr;
DWORD len = file.Read(&hdr, sizeof(hdr));

if ((len!=sizeof(hdr)) || (hdr.bfType!=BITMAP_TYPE))
{
TRACE0("***CDib: bad BITMAPFILEHEADER\n");
return FALSE;
}

len = file.GetLength() - len;
m_pbmih = (BITMAPINFOHEADER*)new char[len];
file.Read(m_pbmih, len);

return TRUE;
}

BOOL CDib::Draw(CClientDC& dc, const CRect* rcDst, const CRect* rcSrc)
{
if (!m_pbmih)
return FALSE;

if (!m_hdd)
VERIFY(m_hdd = DrawDibOpen());

CRect rc;

if (!rcSrc)
{
// if no source rect, use whole bitmap
rc.SetRect(0, 0, m_pbmih->biWidth, m_pbmih->biHeight);
rcSrc=&rc;
}

if (!rcDst)
{
// if no destination rect, use source
rcDst=rcSrc;
}
```

```
// This is as easy as it gets in Windows.
return DrawDibDraw(m_hdd, dc, rcDst->left, rcDst->top, rcDst->Width(), rcDst->Height(), m_pbmih,
GetBits(), rcSrc->left, rcSrc->top, rcSrc->Width(), rcSrc->Height(), 0);
}

BOOL CDib::Draw(CClientDC *pDC)
{
if( m_hObject )
{
CPalette* pOldPal = pDC->SelectPalette(CPalette::FromHandle(m_pal),FALSE);

pDC->RealizePalette();

BOOL bRet = DrawBitmap(*pDC, this); // as before

pDC->SelectPalette(pOldPal, TRUE);

return bRet;
}

return FALSE;
}

BOOL CDib::CreatePalette()
{
int i = 0;
int nColors = GetNumPaletteColors();
CDC *pDC = CDC::FromHandle( GetDC(AfxGetApp()->m_pMainWnd->GetSafeHwnd()) );
CDC mdc; // memory DC
RGBQUAD *pRgbQuad = NULL;
LPLOGPALETTE lpPal;
HANDLE hLogPal;
CBitmap* pOld = NULL;

pRgbQuad = new RGBQUAD[nColors];
mdc.CreateCompatibleDC(pDC);
pOld = mdc.SelectObject(this);
GetDIBColorTable( mdc.GetSafeHdc(), 0, nColors, pRgbQuad );

if( nColors )
```

```
{
hLogPal = GlobalAlloc (GHND, sizeof (LOGPALETTE) +
sizeof (PALETTEENTRY) * nColors);
lpPal = (LPLOGPALETTE) GlobalLock (hLogPal);
lpPal->palVersion = 0x300;
lpPal->palNumEntries = nColors;

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

m_pal = ::CreatePalette( lpPal );

GlobalUnlock(hLogPal);
GlobalFree(hLogPal);
}

delete []pRgbQuad;

ReleaseDC( AfxGetApp()->m_pMainWnd->GetSafeHwnd(), pDC->GetSafeHdc() );

return TRUE;
}

BOOL CDib::DrawBitmap(CDC& dc, CBitmap* pbm)
{
	CDC mdc; // memory DC
	BITMAP bm;

	mdc.CreateCompatibleDC(&dc);

	CBitmap* pOld = mdc.SelectObject(pbm);
	pbm->GetObject(sizeof(bm), &bm);

	BOOL bRet = dc.BitBlt( 0, 0, bm.bmWidth, bm.bmHeight, &mdc, 0, 0, SRCCOPY);

	mdc.SelectObject(pOld);
```

```
        return bRet;
}
```

SUBPROGRAM

MYDIBVIEWER

```
// MyDIBViewerDlg.cpp : implementation file
//

#include "stdafx.h"
#include "MyDIBViewer.h"
#include "MyDIBViewerDlg.h"

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

/////////////////////////////////////////////////////////////////////////////
// CMyDIBViewerDlg dialog

CMyDIBViewerDlg::CMyDIBViewerDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CMyDIBViewerDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CMyDIBViewerDlg)
        m_Value = "";
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMyDIBViewerDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CMyDIBViewerDlg)
        DDX_Text(pDX, IDC_EDIT1, m_Value);
        //}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CMyDIBViewerDlg, CDialog)
	//{{AFX_MSG_MAP(CMyDIBViewerDlg)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMyDIBViewerDlg message handlers

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

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	m_Picture.Load("Photo.bmp");


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

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

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

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

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

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
		CClientDC dc(this);
		CRect src,dst;
		//src.SetRect(0,0,100,100);
		//dst.SetRect(0,0,100,100);
		m_Picture.Draw(&dc);

	}

}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CMyDIBViewerDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CMyDIBViewerDlg::OnOK()
{
	// TODO: Add extra validation here
	DWORD red = 0, green = 0, blue = 0;
	COLORREF pix;
	m_Value = "none";
	CClientDC dc(this);

	for (int x = 0; x<=176; x++)
	{
		for (int y = 0; y<=144; y++)
```

```
                {
                        pix = dc.GetPixel(x,y);
                        red += GetRValue(pix);
                        green += GetGValue(pix);
                        blue += GetBValue(pix);
                }
        }
        if((red>green) && (red>blue)) m_Value = "red";
        if((green>red) && (green>blue)) m_Value = "green";
        if((blue>red) && (blue>green)) m_Value = "blue";
        UpdateData(FALSE);
        //CDialog::OnOK();
}
```