ANEXO A: BIBLIOGRAFÍA

A continuación se recoge la bibliografía que se ha empleado para la elaboración de este proyecto.

[I] Programación avanzada con Visual C++ 6.0 / David J. Kruglinski, Scot Wingo, George Shepherd ; Traducción Ricardo de Córdoba Herralde ; revisión técnica Antonio Vaquero Sánchez, Baltasar Férnandez Manión

[II] Interface Control Document for the Miniature Integrated Land Navigation (MILNAV) Vehicle Reference Unit (VRU) for Santa Barbara Programs SIAC - VRU Part Number K600A395-01 Colombia - VRU Part Number K600A355-02

[III] Programación avanzada con Visual C++ / David J. Kruglinski ; traducción, Jesús Fabregat Carrascosa ; revisión técnica, Antonio Vaquero Sánchez, Luis Hernández Yánez

[IV] TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994, ISBN 0-201-63346-9.

[V] Microsoft Developer Network (MSDN) (http://msdn.microsoft.com)

[VI] Win32 Socket Client (http://www.codersource.net/win32_socket_client.html)

[VII] Beginning Winsock Programming - Simple TCP server (http://www.codeproject.com/internet/winsockintro01.asp)

[VIII] CCESocket: a general purpose TCP/UDP socket class for WinCE By Marco Zaratti.(http://www.codeproject.com/ce/CCESocket.asp)

[IX] CSerialCom - A Simple Class for Implementing Serial Communication in Win-9X/2000 By Shibu K V. (http://www.codeproject.com/system/cserialcom.asp)

[X] Serial Communications in Win32. Allen Denver. Microsoft Windows Developer Support

ANEXO B: CÓDIGO FUENTE DEL PROGRAMA VIPER

A continuación se muestra el código fuente de los ficheros más relevantes del proyecto. En el cdrom adjunto se encuentran ambos proyectos (viper y servidor) con todos los ficheros necesarios para la compilación del software.

Fichero: ViperDlg.cpp

```
// ViperDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Viper.h"
#include "ViperDlg.h"
#include "DialogMedida.h"
#include "mensajes.h"
#include "Comunicacion.h"

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

// Variables globales
extern struct datos_sistema configuracion_sistema;
extern int estado_sistema;
unsigned char buffer_global[TAM_COMPACTADA];

// Definición de los mensajes que se transmiten desde la Viper a la IMU
CMensajeTxAcceptConfigurationData AcceptConfigurationData;
CMensajeTxAcceptPointingDeviceBoresight AcceptPointingDeviceBoresight;
CMensajeTxAcceptPosition AcceptPosition;
CMensajeTxEnableZeroVelocityUpdates EnableZeroVelocityUpdates;
CMensajeTxInTravelLock InTravelLock;
CMensajeTxOdometerModeRequest OdometerModeRequest;
CMensajeTxOverrideAlert OverrideAlert;
CMensajeTxRealign Realign;
CMensajeTxResetDistance ResetDistance;
CMensajeTxRestart Restart;
CMensajeTxReturnAttitudeData ReturnAttitudeData;
CMensajeTxReturnBuiltInTest ReturnBuiltInTest;
CMensajeTxReturnConfigurationData ReturnConfigurationData;
CMensajeTxReturnExpandedStatusData ReturnExpandedStatusData;
CMensajeTxReturnExtendedConfigurationData ReturnExtendedConfigurationData;
CMensajeTxReturnPointingDeviceBoresightAngles ReturnPointingDeviceBoresightAngles;
CMensajeTxReturnStatusData ReturnStatusData;
CMensajeTxReturnSurveyQuality ReturnSurveyQuality;
CMensajeTxReturnTravelLockData ReturnTravelLockData;
CMensajeTxReturnVruOrientation ReturnVruOrientation;
CMensajeTxReturnVruRateData ReturnVruRateData;
CMensajeTxShutdown TxShutdown;

// Definición de los mensajes que se reciben desde la IMU en la Viper
CMensajeRxStatusData StatusData;
CMensajeRxAttitudeData AttitudeData;
CMensajeRxBuiltInTest BuiltInTest;
CMensajeRxConfigurationData ConfigurationData;
CMensajeRxExpandedStatusData ExpandedStatusData;
CMensajeRxExtendedConfigurationData ExtendedConfigurationData;
CMensajeRxPointingDeviceBoresightAngles PointingDeviceBoresightAngles;
CMensajeRxSurveyQuality SurveyQuality;
CMensajeRxTravelLockData TravelLockData;
CMensajeRxVruOrientation VruOrientation;
CMensajeRxVruRateData VruRateData;

// Punteros al mensajeTx y al mensajeRx
```

```
CCabeceraTx* pMensajeTx;
CCabeceraRx* pMensajeRx;

// Buffers de transmisión y recepción de mensajes UDP
struct datos_udp bufferTxUdp;
struct datos_udp bufferRxUdp;

// Puntero a la clase socket
CMySocket *m_socket;
CMySocket *m_serviceSocket;
CMySocket *m_socket_comunicacion;

// Prueba de los datos compactados
unsigned char compactada[50];

// Instancia de la clase CSerialCom
CSerialCom SerialPort;

// Buffers de transmisión y recepción de mensajes por el puerto serie
BYTE bufferRxSerie;
BYTE bufferTxSerie;
char bufferIn[512];
char bufferOut[512];

/*********************************************
 - Nombre: comprueba_cabecera
 - Descripción: Esta función se encarga de comprobar mediante la cabecera
 que nos envía la IMU (bytes + codigo) asociar que mensaje hemos recibido
 Además asigna el puntero a la clase recibida
 - Devuelve: Un 1 si todo ha ido perfectamente
*********************************************/
bool comprueba_cabecera(int bytes, int codigo, int secuencia) {

	int status_ok = 0;
	int numero_bytes = 0;

	// Lo primero es mirar el código del mensaje y asignar el puntero a la clase
recibida
	switch(codigo)
	{
		// STATUS DATA
	case 0:
		numero_bytes = 4;
		pMensajeRx = &StatusData;
		break;

		// TRAVEL LOCK DATA
	case 3:
		numero_bytes = 16;
		pMensajeRx = &TravelLockData;
		break;

		// ATTITUDE DATA
	case 2:
		numero_bytes = 16;
		pMensajeRx = &AttitudeData;
		break;

		// BUILT-IN-TEST DATA
	case 4:
		numero_bytes = 6;
		pMensajeRx = &BuiltInTest;
		break;

		// EXTENDED CONFIGURATION DATA
	case 6:
		numero_bytes = 34;
		pMensajeRx = &ExtendedConfigurationData;
		break;

		// VRU ORIENTATION
	case 10:
		numero_bytes = 16;
		pMensajeRx = &VruOrientation;
		break;
```

```
                // VRU RATE DATA
        case 11:
                numero_bytes = 10;
                pMensajeRx = &VruRateData;
                break;

                // CONFIGURATION DATA
        case 18:
                numero_bytes = 37;
                pMensajeRx = &ConfigurationData;
                break;

                // POINTING DEVICE BORESIGHT ANGLES
        case 20:
                numero_bytes = 10;
                pMensajeRx = &PointingDeviceBoresightAngles;
                break;

                // SURVEY QUALITY
        case 21:
                numero_bytes = 10;
                pMensajeRx = &SurveyQuality;
                break;

                // EXPANDED STATUS DATA
        case 31:
                numero_bytes = 6;
                pMensajeRx = &ExpandedStatusData;
                break;
        // Si es cualquier otro valor es que es una cabecera errónea
        default:
                status_ok = 0;
                break;
        }

        // Ahora comprobamos que la longitud del paquete es correcta
        if ( numero_bytes == bytes )
        {
                status_ok = 1;
        }
        else
                status_ok = 0;

        return status_ok;

}

/////////////////////////////////////////////////////////////////////////////
// CViperDlg dialog

CViperDlg::CViperDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CViperDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CViperDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CViperDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CViperDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CViperDlg, CDialog)
        //{{AFX_MSG_MAP(CViperDlg)
        ON_BN_CLICKED(IDC_MEDIDA, OnMedida)
        ON_WM_TIMER()
        ON_MESSAGE(ON_RECEIVE, OnReceive)
        ON_MESSAGE(ON_ACCEPT, OnAccept)
        ON_BN_CLICKED(IDC_COMUNICACION, OnComunicacion)
        //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CViperDlg message handlers

BOOL CViperDlg::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

	CenterWindow(GetDesktopWindow());	// center to the hpc screen

	// TODO: Add extra initialization here

	// Leemos la configuración desde un fichero de texto
	if ( lee_fichero("../FlashDisk/conf.txt",&configuracion_sistema) )
	{
		// Colocamos en el título de la ventana si somos Referencia o Móvil
		if (configuracion_sistema.viper_referencia)
			CWnd::SetWindowText(_T("VIPER DE REFERENCIA"));
		else
			CWnd::SetWindowText(_T("VIPER MOVIL"));

		CString puerto = "COM";
		switch(configuracion_sistema.port_serie)
		{
		case 1:
			puerto = puerto + "1:";
			break;
		case 2:
			puerto = puerto + "2:";
			break;
		case 3:
			puerto = puerto + "3:";
			break;
		case 4:
			puerto = puerto + "4:";
			break;
		case 5:
			puerto = puerto + "5:";
			break;
		default: break;
		}

		// Configuración del puerto serie
		SerialPort.OpenPort(puerto);
		SerialPort.ConfigurePort(CBR_9600, 8, true, EVENPARITY , ONESTOPBIT);
		SerialPort.SetCommunicationTimeouts(500,500,500,500,500);

		// Prueba de envío por el puerto serie
		compactada[0] = 3;
		compactada[1] = 10;
		compactada[2] = 20;
		compactada[3] = 30;

		SerialPort.WriteByte(compactada);

		// Inicializamos el timer de lectura del puerto serie cada 5s
		m_nTimerSerie = SetTimer(1, 5000, 0);

		if (!m_nTimerSerie)
			AfxMessageBox(_T("ERROR TIMER"));

		// Creación de la clase CMySocket y conexión con el PC
		// Un socket es para la conexión UDP y el otro para la TCP
		m_socket = new CMySocket(this);
		m_socket_comunicacion = new CMySocket(this);
		m_socket_comunicacion->Create(SOCK_STREAM);
		m_socket_comunicacion->Accept(PUERTO_TCP_RESULTADOS);

		// Comprobamos que podemos conectarnos al equipo
		if (m_socket->Create(SOCK_DGRAM) == TRUE)
```

```
		{
			// Realizamos la conexión. Esta conexión es virtual ya que estamos
			// usando UDP y este protocolo es connectionless. Realizamos esto
			// para guardar los datos del ordenador remoto.
			if (m_socket-
>Connect(configuracion_sistema.ip_viper_remota,configuracion_sistema.port_udp) )
			{
				// Prueba de envío de datos
				bufferTxUdp.cabecera = 1;
				bufferTxUdp.datos_enteros = 2;
				bufferTxUdp.datos_flotantes = 3.4;

				m_socket->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));

				// Actualizamos el estado del sistema
				estado_sistema = ESTADO_REPOSO;
			}
			else
				AfxMessageBox(_T("Error Connect UDP"));
		}
		else
			AfxMessageBox(_T("Error Create UDP"));
	}
	else
		// Si no podemos leer el fichero abortamos el programa
		AfxMessageBox(_T("Error al leer el fichero de configuración"));


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

LRESULT CViperDlg::OnReceive(WPARAM wParam, LPARAM lParam)
{
	// Comprobamos que hay datos en el buffer de entrada
	int len = m_socket->GetDataSize();

	if(len > 0)
	{
		m_socket->Read((char *)(&bufferRxUdp), len);

		// Analizamos la cabecera del paquete que nos ha llegado
		switch(bufferRxUdp.cabecera)
		{
		// Esta cabecera simplemente es para probar la recepción de paquetes
		case 0:
			if (estado_sistema == ESTADO_REPOSO)
				AfxMessageBox(_T("INCOMING"));
			break;

		case MENSAJE_UDP_ALINEACION:
			// Este caso es cuando estamos sincronizando ambas Viper
			// Para ello debemos estar en el estado de reposo
			if (estado_sistema == ESTADO_ESPERANDO_SINCRONIZACION)
			{
				CButton* p_inicio = (CButton *)m_pDialogMedida-
>CWnd::GetDlgItem(IDC_INICIO_TRAYECTORIA);

				estado_sistema = ESTADO_TRAYECTORIA;
				AfxMessageBox(_T("IMUs SINCRONIZADAS."));

				p_inicio->EnableWindow(TRUE);
				m_pDialogMedida->CWnd::SetWindowText(_T("TRAYECTORIA"));
			}
			break;
		case MENSAJE_UDP_FIN_TRAYECTORIA:
			// Ha llegado el mensaje de fin de trayectoria
			// Para ello debemos estar en el estado de trayectoria
			if (estado_sistema == ESTADO_TRAYECTORIA)
			{
				AfxMessageBox(_T("FIN DE TRAYECTORIA"));
				m_pDialogMedida->CWnd::SetWindowText(_T("FIN DE
TRAYECTORIA"));
			}
			break;
		default:
			break;
```

```
               }

       }

       return 0;
}

LRESULT CViperDlg::OnAccept(WPARAM wParam, LPARAM lParam)
{
       //AfxMessageBox(_T("Ha llegado una conexión TCP"));

       m_serviceSocket = new CMySocket(this);
    m_serviceSocket->AcceptServiceSocket((SOCKET) wParam);

       // return value
    BOOL bRet = TRUE;
    // used to monitor the progress of a sending operation
    int fileLength, cbLeftToSend;
    // pointer to buffer for sending data
    // (memory is allocated after sending file size)
    BYTE* sendData = NULL;

    CFile sourceFile;
    CFileException fe;
    BOOL bFileIsOpen = FALSE;

       if( !( bFileIsOpen = sourceFile.Open( _T("../FlashDisk/resultados.csv"),
          CFile::modeRead | CFile::typeBinary, &fe ) ) )
    {
        TCHAR strCause[256];
        fe.GetErrorMessage( strCause, 255 );
              AfxMessageBox(strCause);
        /* you should handle the error here */

        bRet = FALSE;
        goto PreReturnCleanup;
    }

    // first send length of file

    fileLength = sourceFile.GetLength();
    fileLength = htonl( fileLength );

    cbLeftToSend = sizeof( fileLength );

    do
    {
        int cbBytesSent;
        BYTE* bp = (BYTE*)(&fileLength) + sizeof(fileLength) - cbLeftToSend;
        cbBytesSent = m_serviceSocket->Send( (char *)bp, cbLeftToSend );

        // test for errors and get out if they occurred
        if ( cbBytesSent == SOCKET_ERROR )
        {
            int iErr = ::GetLastError();
                      AfxMessageBox(_T("ERROR 2"));
            bRet = FALSE;
            goto PreReturnCleanup;
        }

        // data was successfully sent, so account
        // for it with already-sent data
        cbLeftToSend -= cbBytesSent;
    }
    while ( cbLeftToSend>0 );


    // now send the file's data
    sendData = new BYTE[SEND_BUFFER_SIZE];

    cbLeftToSend = sourceFile.GetLength();

    do
    {
        // read next chunk of SEND_BUFFER_SIZE bytes from file
```

```
        int sendThisTime, doneSoFar, buffOffset;

        sendThisTime = sourceFile.Read( sendData, SEND_BUFFER_SIZE );
        buffOffset = 0;

        do
        {
            doneSoFar = m_serviceSocket->Send( (char *)sendData + buffOffset,
                                                sendThisTime );

            // test for errors and get out if they occurred
            if ( doneSoFar == SOCKET_ERROR )
            {
                int iErr = ::GetLastError();

                /* you should handle the error here */

                bRet = FALSE;
                goto PreReturnCleanup;
            }

            // data was successfully sent,
            // so account for it with already-sent data

            buffOffset += doneSoFar;
            sendThisTime -= doneSoFar;
            cbLeftToSend -= doneSoFar;
        }
        while ( sendThisTime > 0 );

    }
    while ( cbLeftToSend > 0 );


        PreReturnCleanup: // labelled goto destination

    // free allocated memory
    // if we got here from a goto that skipped allocation,
    // delete of NULL pointer
    // is permissible under C++ standard and is harmless
    delete[] sendData;

    if ( bFileIsOpen )
        sourceFile.Close();
        // only close file if it's open (open might have failed above)


    if (bRet)
            AfxMessageBox(_T("Resultados enviados"));
        else
            AfxMessageBox(_T("Fallo en el envío"));


    return 0;
}
void CViperDlg::OnMedida()
{
        // TODO: Add your control notification handler code here
        m_pDialogMedida = new CDialogMedida(this);
        m_pDialogMedida->DoModal();

        //CDialogMedida dlg(this);
        //dlg.DoModal();


}

int CViperDlg::lee_fichero(LPCSTR nombre_fichero, struct datos_sistema *
configuracion_sistema)
{
        bool fSuccess = TRUE;

        // Descriptor de fichero
        FILE *fichero = fopen(nombre_fichero,"rb");

        // Buffer donde almacenamos cada línea leída
```

```
	char linea_leida[CARACTERES_LINEA];

	// Comprobamos que el fichero se ha podido crear
	if (fichero != NULL)
	{
		// Recorremos el fichero entero hasta llegar al final
		while (!feof(fichero))
		{
			// Leemos línea por línea
			fgets(linea_leida,CARACTERES_LINEA,fichero);

			// Una vez que tenemos la línea leida leemos la configuración
			if (!analiza_linea(linea_leida,configuracion_sistema))
				fSuccess = FALSE;
		}

	}
	else
		fSuccess = FALSE;

	// Cerramos el fichero
	fclose(fichero);

	return fSuccess;

}

bool CViperDlg::analiza_linea(char *linea_leida, datos_sistema *configuracion_sistema)
{
	bool fSuccess = TRUE;
	char *posicion = NULL;
	char variable[CARACTERES_LINEA];
	int tamano_variable = 0;

	// Intentamos localizar cadenas del tipo: variable=valor
	// strchr me devuelve un puntero a la posición del '='
	posicion = strchr(linea_leida,SIGNO_IGUAL);

	if (posicion)
	{
		tamano_variable = (strlen(linea_leida) - strlen(posicion));
		strncpy(variable,linea_leida,tamano_variable);

		// Ya tengo las cadenas variable y valor. Procedemos a la identificación
		if (strncmp(variable,IP_LOCAL,TAMANO_IP_LOCAL) == 0)
			configuracion_sistema->ip_local = (posicion+1);
		else if (strncmp(variable,IP_VIPER_REMOTA,TAMANO_IP_VIPER_REMOTA ) == 0)
			configuracion_sistema->ip_viper_remota = (posicion+1);
		else if (strncmp(variable,IP_PC,TAMANO_IP_PC ) == 0)
			configuracion_sistema->ip_pc = (posicion+1);
		else if (strncmp(variable,PUERTO_UDP,TAMANO_PUERTO_UDP ) == 0)
			configuracion_sistema->port_udp = atoi(posicion+1);
		else if (strncmp(variable,PUERTO_SERIE,TAMANO_PUERTO_SERIE ) == 0)
			configuracion_sistema->port_serie = atoi(posicion+4);
		else if (strncmp(variable,VIPER_REFERENCIA,TAMANO_VIPER_REFERENCIA ) ==
0)
			configuracion_sistema->viper_referencia = atoi(posicion+1);
		else if (strncmp(variable,MODO_TRABAJO,TAMANO_MODO_TRABAJO ) == 0)
			configuracion_sistema->modo_trabajo = atoi(posicion+1);
		else
			fSuccess = FALSE;
	}
	else
		fSuccess = FALSE;

	return fSuccess;
}

void CViperDlg::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default

	// Copiamos todo lo que hay en el buffer de rx del puerto serie a bufferIn
	if ( SerialPort.lee(bufferIn) )
	{
		// Una vez que hemos vaciado el buffer de rx de Windows usamos la función
```

```
		// Extract para separar una trama y copiarla a bufferOut.
		if (SerialPort.Extract(bufferIn,bufferOut))
			AfxMessageBox(_T("Trama recibida"));
	}

	CDialog::OnTimer(nIDEvent);
}

void CViperDlg::OnComunicacion()
{
	// TODO: Add your control notification handler code here
	m_pDialogComunicacion = new CComunicacion(this);
	m_pDialogComunicacion->DoModal();


}
```

Fichero: ViperDlg.h

```
// ViperDlg.h : header file
//

#if !defined(AFX_VIPERDLG_H__3AD5F825_471F_4939_B687_B74EA57F1AB6__INCLUDED_)
#define AFX_VIPERDLG_H__3AD5F825_471F_4939_B687_B74EA57F1AB6__INCLUDED_

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

#include "DialogMedida.h"
#include "Comunicacion.h"

// Defines asociados a las comunicaciones con el PC
#define PUERTO_TCP_RESULTADOS 6969

#define SEND_BUFFER_SIZE 4096
#define RECV_BUFFER_SIZE 4096


/*****************************************************
 Descripción: Estructura del mensaje UDP entre el PC y la VIPER
 Datos: Esta estructura posee tres campos
	- cabecera: Para indicar el código de mensaje
	- datos_enteros:
	- datos_flotantes
*****************************************************/
struct datos_udp {
	int cabecera;
	int datos_enteros;
	float datos_flotantes;
};



/*****************************************************
 Descripción: Estructura con la configuración del sistema

*****************************************************/
struct datos_sistema {

	// Vale TRUE cuando somos la Viper de referencia
	bool viper_referencia;

	// Puerto utilizado para la comunicación UDP
	UINT port_udp;

	// Puerto utilizado para la comunicación SERIE
	UINT port_serie;

	// Dirección IP de la otra viper
	CString ip_viper_remota;

	// Dirección IP del PC
	CString ip_pc;
```

```
	// Dirección IP local
	CString ip_local;

	// Modo de trabajo
	UINT modo_trabajo;
};

/*************************
	FICHERO DE TEXTO
*************************/
#define CARACTERES_LINEA 100
#define SIGNO_IGUAL 61

#define IP_LOCAL "ip_local"
#define IP_VIPER_REMOTA "ip_viper_remota"
#define IP_PC "ip_pc"
#define PUERTO_UDP "puerto_udp"
#define PUERTO_SERIE "puerto_serie"
#define VIPER_REFERENCIA "viper_referencia"
#define MODO_TRABAJO "modo_trabajo"
#define MODO_REAL 0
#define MODO_DEPURACION1 1
#define MODO_DEPURACION2 2

#define TAMANO_IP_LOCAL 8
#define TAMANO_IP_VIPER_REMOTA 15
#define TAMANO_IP_PC 5
#define TAMANO_PUERTO_UDP 10
#define TAMANO_PUERTO_SERIE 12
#define TAMANO_VIPER_REFERENCIA 16
#define TAMANO_MODO_TRABAJO 12

/*********************************************
 Máquina de estados asociados a los mensajes UDP
*********************************************/
#define ESTADO_REPOSO 0
#define ESTADO_ESPERANDO_SINCRONIZACION 1
#define ESTADO_IMUS_PREPARADAS 2
#define ESTADO_TRAYECTORIA 3

/***************************
 Código mensajes UDP
***************************/
#define MENSAJE_UDP_ALINEACION 100
#define MENSAJE_UDP_FIN_TRAYECTORIA 200

/*************************
 Código mensajes IMU
*************************/
#define MENSAJE_IMU_ALINEACION 100

/////////////////////////////////////////////////////////////////////////////
// CViperDlg dialog

#include "MySocket.h"
#include "SerialCom.h"

class CDialogMedida;



class CViperDlg : public CDialog
{
// Construction
public:

	// Variable asociada al timer de lectura del puerto serie
	int m_nTimerSerie;
	bool analiza_linea(char *linea_leida, datos_sistema *configuracion);
	int lee_fichero(LPCSTR nombre_fichero, struct datos_sistema * configuracion);

	CViperDlg(CWnd* pParent = NULL);     // standard constructor
	afx_msg LRESULT OnReceive(WPARAM wParam, LPARAM lParam);
	afx_msg LRESULT OnAccept(WPARAM wParam, LPARAM lParam);
```

```
        // Prueba de ficheros
        FILE *fichero;

        // Dialog Data
        //{{AFX_DATA(CViperDlg)
        enum { IDD = IDD_VIPER_DIALOG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

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

// Implementation
protected:
        CDialogMedida  *  m_pDialogMedida ;
        CComunicacion  *  m_pDialogComunicacion ;
        HICON m_hIcon;

        // Puntero al cuadro de diálogo

        // Generated message map functions
        //{{AFX_MSG(CViperDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnMedida();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnComunicacion();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_VIPERDLG_H__3AD5F825_471F_4939_B687_B74EA57F1AB6__INCLUDED_)
```

Fichero: Comunicación.cpp

```
// Comunicacion.cpp : implementation file
//

#include "stdafx.h"
#include "Viper.h"
#include "Comunicacion.h"
#include "ViperDlg.h"

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

// Defines asociados a las comunicaciones con el PC
#define PUERTO_TCP_CONFIGURACION 7070

#define SEND_BUFFER_SIZE 4096
#define RECV_BUFFER_SIZE 4096

// Variables globales
extern struct datos_sistema configuracion_sistema;
extern CMySocket *m_socket_comunicacion;

/////////////////////////////////////////////////////////////////////////////
// CComunicacion dialog


CComunicacion::CComunicacion(CWnd* pParent /*=NULL*/)
        : CDialog(CComunicacion::IDD, pParent)
{
        //{{AFX_DATA_INIT(CComunicacion)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
```

```
}


void CComunicacion::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CComunicacion)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CComunicacion, CDialog)
	//{{AFX_MSG_MAP(CComunicacion)
	ON_BN_CLICKED(IDC_Recibir, OnRecibir)
	ON_BN_CLICKED(IDC_Resultados, OnResultados)
	ON_BN_CLICKED(IDC_Recibir_Experimentos, OnRecibirExperimentos)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CComunicacion message handlers

void CComunicacion::OnRecibir()
{
	// Creamos el socket y nos conectamos
	CMySocket *m_socket_configuracion;
	m_socket_configuracion = new CMySocket(this);
	m_socket_configuracion->Create(SOCK_STREAM, 4096);

	CString ip_pc  = configuracion_sistema.ip_pc;

	m_socket_configuracion->Connect(ip_pc, PUERTO_TCP_CONFIGURACION);

	// Variable de retorno
    BOOL bRet = TRUE;

	// Variables utilizadas en la operación de recepción
    int dataLength, cbBytesRet, cbLeftToReceive;

	// Puntero al buffer de recepción
    BYTE* recdData = NULL;

	// Fichero donde se almacenan los datos recibidos
    CFile destFile;
    CFileException fe;
    BOOL bFileIsOpen = FALSE;

    // Abrimos el fichero de recepción
	if( !( bFileIsOpen = destFile.Open( _T("../FlashDisk/conf.txt"),
CFile::modeCreate |
	    CFile::modeWrite | CFile::typeBinary, &fe ) ) )
    {
        TCHAR strCause[256];
        fe.GetErrorMessage( strCause, 255 );
        AfxMessageBox(_T("ERROR CON EL NOMBRE DEL FICHERO"));
        /* Manejamos el error */

        bRet = FALSE;
	       // Saltamos al final de la función
        goto PreReturnCleanup;
    }

    // Lo primero que recibimos es el tamaño del fichero
    cbLeftToReceive = sizeof( dataLength );

	// Bucle de recepción de bytes mientras haya bytes que recibir
    do
    {
        BYTE* bp = (BYTE*)(&dataLength) + sizeof(dataLength) - cbLeftToReceive;
        cbBytesRet = m_socket_configuracion->Read( (char *)bp, cbLeftToReceive );

        // Comprobamos errores
        if ( cbBytesRet == SOCKET_ERROR || cbBytesRet == 0 )
        {
            int iErr = ::GetLastError();
```

```
            //AfxMessageBox(_T("ERROR DE RECEPCION DE LONGITUD"));
            /* Manejamos el error */

            //bRet = FALSE;
            //goto PreReturnCleanup;
        }

               // Decrementamos los bytes a recibir
        cbLeftToReceive -= cbBytesRet;
    }
    while ( cbLeftToReceive > 0 );

    dataLength = ntohl( dataLength );

    // Ahora recibimos tramas del fichero y la almacenamos en el buffer
    recdData = new byte[RECV_BUFFER_SIZE];
    cbLeftToReceive = dataLength;

       // Nuevo bucle mientras haya datos que recibir
    do
    {
        int iiGet, iiRecd;

        iiGet = (cbLeftToReceive<RECV_BUFFER_SIZE) ?
                        cbLeftToReceive : RECV_BUFFER_SIZE ;
        iiRecd = m_socket_configuracion->Read( (char *)recdData, iiGet );

        // Comprobamos errores
        if ( iiRecd == SOCKET_ERROR || iiRecd == 0 )
        {
            int iErr = ::GetLastError();
            AfxMessageBox(_T("ERROR A LA HORA DE RECIBIR EL FICHERO"));
            /* Manejamos el error */

            bRet = FALSE;
            goto PreReturnCleanup;
        }

        destFile.Write( recdData, iiRecd); // Write it
        cbLeftToReceive -= iiRecd;

    }
    while ( cbLeftToReceive > 0 );

       PreReturnCleanup: // labelled "goto" destination

       // Borramos la memoria utilizado, cerramos socket y fichero
    delete[] recdData;

    if ( bFileIsOpen )
        destFile.Close();


       if (bRet)
              AfxMessageBox(_T("Configuración recibida. Reinicie el software"));
       else
              AfxMessageBox(_T("Error al recibir configuración"));

       m_socket_configuracion->Disconnect();
}

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

       // TODO: Add extra initialization here
       //AfxMessageBox(_T("PRUEBA"));

       return TRUE;  // return TRUE unless you set the focus to a control
                     // EXCEPTION: OCX Property Pages should return FALSE
}

void CComunicacion::OnResultados()
{
       // Nos ponemos a la escucha de la conexión TCP. Cuando llegue una conexión
       // TCP, saltará el evento OnAccept
```

```
}


void CComunicacion::OnOK()
{
	// TODO: Add extra validation here

	CDialog::OnOK();
}

void CComunicacion::OnRecibirExperimentos()
{
	// Creamos el socket y nos conectamos
	CMySocket *m_socket_configuracion;
	m_socket_configuracion = new CMySocket(this);
	m_socket_configuracion->Create(SOCK_STREAM, 4096);

	CString ip_pc  = configuracion_sistema.ip_pc;

	m_socket_configuracion->Connect(ip_pc, PUERTO_TCP_CONFIGURACION);

	// Variable de retorno
    BOOL bRet = TRUE;

	// Variables utilizadas en la operación de recepción
    int dataLength, cbBytesRet, cbLeftToReceive;

	// Puntero al buffer de recepción
    BYTE* recdData = NULL;

	// Fichero donde se almacenan los datos recibidos
    CFile destFile;
    CFileException fe;
    BOOL bFileIsOpen = FALSE;

    // Abrimos el fichero de recepción
	if( !( bFileIsOpen = destFile.Open( _T("../FlashDisk/experimentos.txt"),
CFile::modeCreate |
	    CFile::modeWrite | CFile::typeBinary, &fe ) ) )
    {
        TCHAR strCause[256];
        fe.GetErrorMessage( strCause, 255 );
        AfxMessageBox(_T("ERROR CON EL NOMBRE DEL FICHERO"));
        /* Manejamos el error */

        bRet = FALSE;
               // Saltamos al final de la función
        goto PreReturnCleanup;
    }

    // Lo primero que recibimos es el tamaño del fichero
    cbLeftToReceive = sizeof( dataLength );

	// Bucle de recepción de bytes mientras haya bytes que recibir
    do
    {
        BYTE* bp = (BYTE*)(&dataLength) + sizeof(dataLength) - cbLeftToReceive;
        cbBytesRet = m_socket_configuracion->Read( (char *)bp, cbLeftToReceive );

        // Comprobamos errores
        if ( cbBytesRet == SOCKET_ERROR || cbBytesRet == 0 )
        {
            int iErr = ::GetLastError();
            //AfxMessageBox(_T("ERROR DE RECEPCION DE LONGITUD"));
            /* Manejamos el error */

            //bRet = FALSE;
            //goto PreReturnCleanup;
        }

               // Decrementamos los bytes a recibir
```

```
        cbLeftToReceive -= cbBytesRet;
    }
    while ( cbLeftToReceive > 0 );

    dataLength = ntohl( dataLength );

    // Ahora recibimos tramas del fichero y la almacenamos en el buffer
    recdData = new byte[RECV_BUFFER_SIZE];
    cbLeftToReceive = dataLength;

       // Nuevo bucle mientras haya datos que recibir
    do
    {
        int iiGet, iiRecd;

        iiGet = (cbLeftToReceive<RECV_BUFFER_SIZE) ?
                    cbLeftToReceive : RECV_BUFFER_SIZE ;
        iiRecd = m_socket_configuracion->Read( (char *)recdData, iiGet );

        // Comprobamos errores
        if ( iiRecd == SOCKET_ERROR || iiRecd == 0 )
        {
            int iErr = ::GetLastError();
            AfxMessageBox(_T("ERROR A LA HORA DE RECIBIR EL FICHERO"));
            /* Manejamos el error */

            bRet = FALSE;
            goto PreReturnCleanup;
        }

        destFile.Write( recdData, iiRecd); // Write it
        cbLeftToReceive -= iiRecd;

    }
    while ( cbLeftToReceive > 0 );

       PreReturnCleanup: // labelled "goto" destination

       // Borramos la memoria utilizado, cerramos socket y fichero
    delete[] recdData;

    if ( bFileIsOpen )
        destFile.Close();


       if (bRet)
              AfxMessageBox(_T("Configuración recibida. Reinicie el software"));
       else
              AfxMessageBox(_T("Error al recibir configuración"));

       m_socket_configuracion->Disconnect();
}
```

Fichero: DialogMedida.cpp

```
// DialogMedida.cpp : implementation file
//

#include "stdafx.h"
#include "Viper.h"
#include "DialogMedida.h"


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

// Variables globales
extern struct datos_sistema configuracion_sistema;
extern int estado_sistema;
extern struct datos_udp bufferTxUdp;
extern struct datos_udp bufferRxUdp;
```

```
extern CMySocket *m_socket;
extern CMySocket *m_serviceSocket;
extern CMySocket *m_socket_comunicacion;

extern unsigned char compactada[50];

extern CSerialCom SerialPort;

extern BYTE bufferRxSerie;
extern BYTE bufferTxSerie;
extern char bufferIn[512];
extern char bufferOut[512];


/////////////////////////////////////////////////////////////////////////////
// CDialogMedida dialog


CDialogMedida::CDialogMedida(CWnd* pParent /*=NULL*/)
	: CDialog(CDialogMedida::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialogMedida)
	//}}AFX_DATA_INIT
}


void CDialogMedida::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialogMedida)
	DDX_Control(pDX, IDC_TIPO_EXPERIMENTO, m_combo_experimento);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialogMedida, CDialog)
	//{{AFX_MSG_MAP(CDialogMedida)
	ON_BN_CLICKED(IDC_IMU_ALINEADA, OnImuAlineada)
	ON_BN_CLICKED(IDC_INICIO_TRAYECTORIA, OnInicioTrayectoria)
	ON_BN_CLICKED(IDC_CANCEL, OnCancel)
	ON_BN_CLICKED(IDC_FIN_TRAYECTORIA, OnFinTrayectoria)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialogMedida message handlers

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

	// TODO: Add extra initialization here

	CButton* pbutton = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
	CButton* p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
	CButton* p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
	CButton* p_experimento = (CButton *)GetDlgItem(IDC_STATIC_EXPERIMENTO);
	CButton* p_pulse_imus = (CButton *)GetDlgItem(IDC_STATIC_PULSE_IMUS);

	CButton* p_combo_experimento = (CButton *)GetDlgItem(IDC_TIPO_EXPERIMENTO);

	// En función de ser la vipe de referencia o la móvil mostramos una interfaz u
otra
	if (configuracion_sistema.viper_referencia)
	{
		// Ocultamos los botones innecesarios
		pbutton->ShowWindow(FALSE);
		p_inicio->ShowWindow(FALSE);
		p_fin->ShowWindow(FALSE);
		p_experimento->ShowWindow(FALSE);
		p_pulse_imus->ShowWindow(FALSE);
		p_combo_experimento->ShowWindow(FALSE);

		// Alineamos nuestra IMU
		compactada[0] = MENSAJE_IMU_ALINEACION;
		compactada[1] = 10;
```

```
		compactada[2] = 20;
		compactada[3] = 30;
		SerialPort.WriteByte(compactada);

		// Cambiamos nuestro estado
		estado_sistema = ESTADO_ESPERANDO_SINCRONIZACION;

		// Esperamos a que la otra Viper nos envíe el mensaje de preparado
		// La función OnReceive procesará este mensaje y activará la bandera
imus_preparadas
		CWnd::SetWindowText(_T("ESPERANDO SINCRONIZACION"));
	}
	else
	{
		// Rellenamos la lista de experimentos a partir del fichero
		if (!lee_experimentos("../FlashDisk/experimentos.txt"))
			AfxMessageBox(_T("Error al leer el fichero de experimentos"));

		p_inicio->EnableWindow(FALSE);
		p_fin->EnableWindow(FALSE);
		m_combo_experimento.SetCurSel(0);
	}

	return TRUE;  // return TRUE unless you set the focus to a control
			// EXCEPTION: OCX Property Pages should return FALSE
}

void CDialogMedida::OnImuAlineada()
{
	// TODO: Add your control notification handler code here
	CButton* pbutton = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
	CButton* p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
	CButton* p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);

	// Puede que esté activado el modo de depuración o no
	if (configuracion_sistema.modo_trabajo == MODO_REAL)
	{
		// Enviamos el paquete de alineación a la IMU
		compactada[0] = MENSAJE_IMU_ALINEACION;
		compactada[1] = 10;
		compactada[2] = 20;
		compactada[3] = 30;
		SerialPort.WriteByte(compactada);

	}
	else if (configuracion_sistema.modo_trabajo == MODO_DEPURACION1)
	{
		p_inicio->EnableWindow(TRUE);
		p_fin->EnableWindow(TRUE);
	}

	// Cambiamos nuestro estado
	estado_sistema = ESTADO_ESPERANDO_SINCRONIZACION;

	// Esperamos a que la otra Viper nos envíe el mensaje de preparado
	// La función OnReceive procesará este mensaje y activará la bandera
imus_preparadas
	CWnd::SetWindowText(_T("ESPERANDO ARRANQUE DE IMUs"));
	pbutton->EnableWindow(FALSE);
}

void CDialogMedida::OnInicioTrayectoria()
{
	// TODO: Add your control notification handler code here
	CButton *p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
	CButton *p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);

	UpdateData(TRUE);

	//m_descripcion_experimento = (CString)m_tipo_experimento;

	p_inicio->EnableWindow(FALSE);
	p_fin->EnableWindow(TRUE);

	CWnd::SetWindowText(_T("TRAYECTORIA"));
```

```
	UpdateData(FALSE);
}


bool CDialogMedida::lee_experimentos(LPCSTR nombre_fichero)
{
	UpdateData(TRUE);
	bool fSuccess = FALSE;
	CString cadena;

	// Descriptor de fichero
	FILE *fichero = fopen(nombre_fichero,"rb");

	// Buffer donde almacenamos cada línea leída
	char linea_leida[CARACTERES_LINEA];

	// Reseteamos la lista del combobox
	m_combo_experimento.ResetContent();

	// Comprobamos que el fichero se ha podido abrir
	if (fichero != NULL)
	{
		// Recorremos el fichero entero hasta llegar al final
		while (!feof(fichero))
		{
			// Leemos línea por línea
			fgets(linea_leida,CARACTERES_LINEA,fichero);

			// Lo metemos en un CString
			cadena=linea_leida;

			// Una vez que tenemos la línea leida añadimos la entrada al
combobox
			m_combo_experimento.AddString(cadena);

			fSuccess = TRUE;
		}

	}
	else
	{
		AfxMessageBox(_T("Fallo con el fichero"));
		fSuccess = FALSE;
	}

	// Cerramos el fichero
	fclose(fichero);

	UpdateData(FALSE);

	return fSuccess;
}

void CDialogMedida::OnCancel()
{
	// TODO: Add your control notification handler code here
	OnOK();

}

void CDialogMedida::OnFinTrayectoria()
{
	// TODO: Add your control notification handler code here

	// Enviamos paquete a la IMU y a la otra VIPER diciendo que ha finalizado la
trayectoria
	if (configuracion_sistema.modo_trabajo == MODO_REAL)
	{
		bufferTxUdp.cabecera = MENSAJE_UDP_FIN_TRAYECTORIA;
		bufferTxUdp.datos_enteros = 2;
		bufferTxUdp.datos_flotantes = 3.4;
		m_socket->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));
	}

	// Realizamos las operaciones de cálculo implementando la fórmula
```

```
	// Guardamos los datos del experimento
	experimento.hora = CTime::GetCurrentTime();
	experimento.indice = m_combo_experimento.GetCurSel() + 1; // Ya que el primer
elemento del combo es 0

	int i = 0;
	for (i=0;i<9;i++)
	{
		experimento.datos[i] = (i + 0.2344);
	}

	// Volcamos los datos en el fichero de texto
	if (escribe_resultados("../FlashDisk/resultados.csv", experimento))
		AfxMessageBox(_T("Resultados almacenados correctamente"));
	else
		AfxMessageBox(_T("Error al almacenar los resultados"));

	CButton* p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
	p_fin->EnableWindow(FALSE);

}

// Esta función se encarga de pasar los datos de la estructura resultado_experimento
// a un fichero de texto en el que los datos se separan por ';'
// Cada vez que se llama a la función se escribe una nueva línea en el fichero
bool CDialogMedida::escribe_resultados(LPCSTR nombre_fichero, struct
resultado_experimento)
{
	bool resultado = TRUE;

	// Descriptor de fichero
	FILE *fichero = fopen(nombre_fichero,"a+");

	fprintf(fichero,"%d/%d/%d %d:%d:%d;%d;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;\n",
experimento.hora.GetDay(), experimento.hora.GetMonth(), experimento.hora.GetYear(),
experimento.hora.GetHour(), experimento.hora.GetMinute(), experimento.hora.GetSecond(),
experimento.indice, experimento.datos[0], experimento.datos[1], experimento.datos[2],
experimento.datos[3], experimento.datos[4], experimento.datos[5], experimento.datos[6],
experimento.datos[7], experimento.datos[8], experimento.datos[9]);


	fclose(fichero);
	return resultado;

}
```

Fichero: DialogMedida.h

```
#if !defined(AFX_DIALOGMEDIDA_H__194C31DA_5F9C_41C6_BAC9_5804A2BFC573__INCLUDED_)
#define AFX_DIALOGMEDIDA_H__194C31DA_5F9C_41C6_BAC9_5804A2BFC573__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DialogMedida.h : header file
//
#include "ViperDlg.h"

/////////////////////////////////////////////////////
 Descripción: Estructura utilizada para almacenar los resultados
 tras las medidias. Luego una función se encargará de coger estos
 datos y volcarlos en un fichero csv.

 Campos:
	- Fecha
	- Hora
	- Tipo de experimeto (índice)
	- 10 números flotantes

/////////////////////////////////////////////////////*/
struct resultado_experimento {

	// Fecha y hora del experimento
	CTime hora;
```

```
        // Índice del experimento
        UINT indice;

        // Un array de 10 datos flotantes
        float datos[10];
};

/////////////////////////////////////////////////////////////////////////////
// CDialogMedida dialog

class CDialogMedida : public CDialog
{
// Construction
public:
        struct resultado_experimento experimento;
        bool escribe_resultados(LPCSTR nombre_fichero, struct resultado_experimento);
        bool lee_experimentos(LPCSTR nombre_fichero);
        CDialogMedida(CWnd* pParent = NULL);   // standard constructor
// Dialog Data
        //{{AFX_DATA(CDialogMedida)
        enum { IDD = IDD_MEDIDA };
        CComboBox      m_combo_experimento;
        //}}AFX_DATA


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

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDialogMedida)
        virtual BOOL OnInitDialog();
        afx_msg void OnImuAlineada();
        afx_msg void OnInicioTrayectoria();
        afx_msg void OnCancel();
        afx_msg void OnFinTrayectoria();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOGMEDIDA_H__194C31DA_5F9C_41C6_BAC9_5804A2BFC573__INCLUDED_)
```

Fichero: mensajes.cpp

```
// mensajes.cpp: implementation of the CCabeceraTx class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Viper.h"
#include "mensajes.h"

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

/*************************************
 Funcion: bucle_compactar
 Descripción:  Esta función se encarga de realizar los desplazamientos de bits
                              para compactar los datos en memoria. Le pasas como
parámetro un número
                              y la función lo encaja entre byte_min y byte_max
 Parámetros:
```

```
        - numero: Es el digito que queremos compactar en memoria
        - byte_min: Byte inferior de la esctructura compactar (Sin contar los 2 bytes de
cabecera)
        - byte_max: Byte superior de la esctructura compactar (Sin contar los 2 bytes de
cabecera)
************************************/
void CCabeceraTx::bucle_compactar(int numero, int byte_min ,int byte_max) {

        int numero_byte = 0;
        int byte = 0;
        int bit = 0;
        int mascara = 0;
        int resto = 0;

        numero_byte = 0;
        for (byte = (byte_max+1);byte >= (byte_min+1); byte --)
        {
                for (bit = 0;bit < 8; bit ++)
                {
                        mascara = (1 << (numero_byte*8+bit));
                        resto = numero & mascara;

                        if (resto)
                        {
                                compactada[byte] = compactada[byte] | (1 << bit);
                        }
                }
                numero_byte ++;
        }
}


/****************************************
 Función: compactar()
 Clase: CMensajeTxAcceptPosition
 Descripción:  Esta función coloca los valores de la clase
                           en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxAcceptPosition::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
                compactada[i]=0;
        }

        /*********************************************
        **               CABECERA                   **
        ** Nota: Los bytes empiezan desde el 0       **
        *********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);

        /*********************************************
        **            NORTHING (3 bytes)             **
        *********************************************/
        bucle_compactar(northing,1,3);

        /*********************************************
        **            ALTITUDE (2 bytes)             **
        *********************************************/
        bucle_compactar(altitude,4,5);

        /*********************************************
        **         EXTENDED ZONE (1 byte)            **
        *********************************************/
        bucle_compactar(extended_zone,6,6);

        /*********************************************
        **            HEMISPHERE (1 byte)            **
```

```
        ********************************************/
        bucle_compactar(hemisphere,7,7);

        /*********************************************
        **     PRIME SYSTEM SPHEROID (1 byte)         **
        *********************************************/
        bucle_compactar(prime_system_spheroid,8,8);

        /*********************************************
        **             EASTING (1 byte)               **
        *********************************************/
        bucle_compactar(easting,9,11);

}

/****************************************
 Función: compactar()
 Clase: CMensajeTxAcceptPointingDeviceBoresight
 Descripción:  Esta función coloca los valores de la clase
                            en un bloque compacto en memoria (array compactada)
 Detalles:     Los grados que hay que enviarle al IMU se miden en mils.
                       Nosotros en memoria usamos radianes, por lo que hay que hacer la
conversión
****************************************/
void CMensajeTxAcceptPointingDeviceBoresight::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
                compactada[i]=0;
        }

        /*********************************************
        **               CABECERA                    **
        ** Nota: Los bytes empiezan desde el 0        **
        *********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);

        /*********************************************
        **               A (2 bytes)                 **
        *********************************************/
        // En memoria tenemos el valor en radianes. Pasamos a mils con precisión 0.1
        bucle_compactar(a*RAD_MILS*10,1,2);

        /*********************************************
        **               B (2 bytes)                  **
        *********************************************/
        bucle_compactar(b*RAD_MILS*10,3,4);

        /*********************************************
        **               Gamma (1 byte)              **
        *********************************************/
        bucle_compactar(gamma*RAD_MILS*10,5,6);

}

/****************************************
 Función: compactar()
 Clase: CMensajeTxAcceptConfigurationData
 Descripción:  Esta función coloca los valores de la clase
                            en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxAcceptConfigurationData::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
```

```
            compactada[i]=0;
        }

        /*********************************************
        **               CABECERA                   **
        ** Nota: Los bytes empiezan desde el 0       **
        *********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);

        /*********************************************
        **               ALFA (2 bytes)             **
        *********************************************/
        bucle_compactar(alfa*RAD_MILS*10,1,2);

        /*********************************************
        **               BETA (2 bytes)             **
        *********************************************/
        bucle_compactar(beta*RAD_MILS*10,3,4);

        /*********************************************
        **               Gamma (2 bytes)            **
        *********************************************/
        bucle_compactar(gamma*RAD_MILS*10,5,6);

        /*********************************************
        **         ESPACIO CON CEROS (5 bytes)      **
        *********************************************/
        bucle_compactar(0,7,12);

        /*********************************************
        **           INCREMENTO_X (1 byte)          **
        *********************************************/
        bucle_compactar(incremento_x*10,13,13);

        /*********************************************
        **           INCREMENTO_Y (1 byte)          **
        *********************************************/
        bucle_compactar(incremento_y*10,14,14);

        /*********************************************
        **           INCREMENTO_Z (1 byte)          **
        *********************************************/
        bucle_compactar(incremento_z*10,15,15);

        /*********************************************
        **           ZUPT_ZUPT (1 byte)             **
        *********************************************/
        bucle_compactar(zupt_zupt*4,16,16);

        /*********************************************
        **           ZUPT_ODOMETER (1 byte)         **
        *********************************************/
        bucle_compactar(zupt_odometer,17,17);

        /*********************************************
        **           NORMAL_ALIGN_TIME (1 byte)     **
        *********************************************/
        bucle_compactar(normal_align_time*4,18,18);

        /*********************************************
        **     STORED_HEADING_ALIGN_TIME (1 byte)   **
        *********************************************/
        bucle_compactar(stored_heading_align_time*4,19,19);

        /*********************************************
        **           REALIGN TIME (1 byte)          **
        *********************************************/
        bucle_compactar(realign_time*4,20,20);

        /*********************************************
        **           MANUAL_ALIGN_TIME (1 byte)     **
```

```
        ********************************************/
        bucle_compactar(manual_align_time*4,21,21);

        /*********************************************
        **           ESPACIO CON CEROS (1 byte)       **
        *********************************************/
        bucle_compactar(0,22,22);

        /*********************************************
        **           CONFIGURATION_FLAG (2 bytes)     **
        *********************************************/
        bucle_compactar(configuration_flag,23,24);

        /*********************************************
        **               VMS (2 bytes)                **
        *********************************************/
        bucle_compactar(vms*1000,25,26);

        /*********************************************
        **          ESPACIO CON CEROS (4 byte)        **
        *********************************************/
        bucle_compactar(0,27,30);

        /*********************************************
        **           VRU_MATRIX (3 bytes)             **
        *********************************************/
        bucle_compactar(vru_matrix,31,33);
}


/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnVruRateData
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnVruRateData::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
                compactada[i]=0;
        }

        /*********************************************
        **               CABECERA                     **
        ** Nota: Los bytes empiezan desde el 0        **
        *********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);

}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnConfigurationData
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnConfigurationData::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
                compactada[i]=0;
        }

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

```
        **               CABECERA                      **
        ** Nota: Los bytes empiezan desde el 0        **
        ********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);

        /*********************************************
        **             SUBCODE (1 byte)              **
        *********************************************/
        bucle_compactar(7,1,1);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnAttitudeData
 Descripción:  Esta función coloca los valores de la clase
                            en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnAttitudeData::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
                compactada[i]=0;
        }

        /*********************************************
        **               CABECERA                      **
        ** Nota: Los bytes empiezan desde el 0        **
        *********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnBuiltInTest
 Descripción:  Esta función coloca los valores de la clase
                            en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnBuiltInTest::compactar() {

        int i = 0;

        // Ponemos a 0 todo el array de compactada
        for(i=0;i<TAM_COMPACTADA;i++)
        {
                compactada[i]=0;
        }

        /*********************************************
        **               CABECERA                      **
        ** Nota: Los bytes empiezan desde el 0        **
        *********************************************/
        // El primer byte es el número de bytes del mensaje
        compactada[0] = bytes;

        // El segundo es el número de secuencia y el código de mensaje
        compactada[1] = codigo;
        compactada[1] = compactada[1]|(secuencia<<5);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnTravelLockData
 Descripción:  Esta función coloca los valores de la clase
```

```
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnTravelLockData::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/***********************************************
	**               CABECERA                     **
	** Nota: Los bytes empiezan desde el 0        **
	***********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}


/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnVruOrientation
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnVruOrientation::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/***********************************************
	**               CABECERA                     **
	** Nota: Los bytes empiezan desde el 0        **
	***********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnStatusData
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnStatusData::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/***********************************************
	**               CABECERA                     **
	** Nota: Los bytes empiezan desde el 0        **
	***********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;
```

```
	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnPointingDeviceBoresightAngles
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnPointingDeviceBoresightAngles::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**               CABECERA                   **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);

	/*********************************************
	**             SUBCODE (1 byte)             **
	*********************************************/
	bucle_compactar(8,1,1);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnExpandedStatusData
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnExpandedStatusData::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**               CABECERA                   **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);

	/*********************************************
	**             SUBCODE (1 byte)             **
	*********************************************/
	bucle_compactar(9,1,1);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnSurveyQuality
 Descripción:  Esta función coloca los valores de la clase
                              en un bloque compacto en memoria (array compactada)
```

```
****************************************/
void CMensajeTxReturnSurveyQuality::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**              CABECERA                    **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);

	/*********************************************
	**            SUBCODE (1 byte)              **
	*********************************************/
	bucle_compactar(10,1,1);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxOverrideAlert
 Descripción:  Esta función coloca los valores de la clase
                          en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxOverrideAlert::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**              CABECERA                    **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/****************************************
 Función: compactar()
 Clase: CMensajeTxReturnExtendedConfigurationData
 Descripción:  Esta función coloca los valores de la clase
                           en un bloque compacto en memoria (array compactada)
****************************************/
void CMensajeTxReturnExtendedConfigurationData::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**              CABECERA                    **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
```

```
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);

	/*********************************************
	**             SUBCODE (1 byte)              **
	*********************************************/
	bucle_compactar(8,1,2);
}

/*****************************************
 Función: compactar()
 Clase: CMensajeTxResetDistance
 Descripción:  Esta función coloca los valores de la clase
                           en un bloque compacto en memoria (array compactada)
*****************************************/
void CMensajeTxResetDistance::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**               CABECERA                   **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/*****************************************
 Función: compactar()
 Clase: CMensajeTxRestart
 Descripción:  Esta función coloca los valores de la clase
                          en un bloque compacto en memoria (array compactada)
*****************************************/
void CMensajeTxRestart::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**               CABECERA                   **
	** Nota: Los bytes empiezan desde el 0       **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/*****************************************
 Función: compactar()
 Clase: CMensajeTxShutdown
 Descripción:  Esta función coloca los valores de la clase
                          en un bloque compacto en memoria (array compactada)
*****************************************/
void CMensajeTxShutdown::compactar() {
```

```
	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**		CABECERA			**
	** Nota: Los bytes empiezan desde el 0	**
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/*************************************
 Función: compactar()
 Clase: CMensajeTxEnableZeroVelocityUpdates
 Descripción:  Esta función coloca los valores de la clase
                          en un bloque compacto en memoria (array compactada)
*************************************/
void CMensajeTxEnableZeroVelocityUpdates::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**		CABECERA			**
	** Nota: Los bytes empiezan desde el 0	**
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}

/*************************************
 Función: compactar()
 Clase: CMensajeTxInTravelLock
 Descripción:  Esta función coloca los valores de la clase
                          en un bloque compacto en memoria (array compactada)
*************************************/
void CMensajeTxInTravelLock::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**		CABECERA			**
	** Nota: Los bytes empiezan desde el 0	**
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
```

```
	/*********************************************
	**             SUBCODE (1 byte)              **
	*********************************************/
	bucle_compactar(5,1,1);
}

/*************************************
 Función: compactar()
 Clase: CMensajeTxRealign
 Descripción:  Esta función coloca los valores de la clase
                           en un bloque compacto en memoria (array compactada)
*************************************/
void CMensajeTxRealign::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**               CABECERA                    **
	** Nota: Los bytes empiezan desde el 0        **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);
}



/*************************************
 Función: compactar()
 Clase: CMensajeTxOdometerModeRequest
 Descripción:  Esta función coloca los valores de la clase
                           en un bloque compacto en memoria (array compactada)
*************************************/
void CMensajeTxOdometerModeRequest::compactar() {

	int i = 0;

	// Ponemos a 0 todo el array de compactada
	for(i=0;i<TAM_COMPACTADA;i++)
	{
		compactada[i]=0;
	}

	/*********************************************
	**               CABECERA                    **
	** Nota: Los bytes empiezan desde el 0        **
	*********************************************/
	// El primer byte es el número de bytes del mensaje
	compactada[0] = bytes;

	// El segundo es el número de secuencia y el código de mensaje
	compactada[1] = codigo;
	compactada[1] = compactada[1]|(secuencia<<5);

	/*********************************************
	**             SUBCODE (1 byte)              **
	*********************************************/
	bucle_compactar(2,1,1);
}



void CCabeceraRx::descompactar(unsigned char *buffer_global) {
	//cout << "Función descompactar virtual" << endl;
}
```

```
/************************************
 Funcion: bucle_descompactar
 Descripción:  Esta función se encarga de leer el valor comprendido entre byte_min
 y byte_max del buffer_recepción_global y devuelve un valor.
 Parámetros:
        - byte_min: Byte inferior de la esctructura compactar (Sin contar los 2 bytes de
cabecera)
        - byte_max: Byte superior de la esctructura compactar (Sin contar los 2 bytes de
cabecera)
        - buffer: Puntero al buffer global donde se encuentran los datos compactados
 Devuelve: El valor comprendido entre byte_min y byte_max
************************************/
int CCabeceraRx::bucle_descompactar(int byte_min ,int byte_max, unsigned char *buffer) {

        int numero_byte = 0;
        int byte = 0;
        int devolver = 0;
        int temporal = 0;

        numero_byte = 0;

        for (byte = (byte_max+3);byte >= (byte_min+3); byte --)
        {
                temporal = buffer[byte];
                devolver = (temporal << numero_byte*8) | devolver;
                numero_byte ++;
        }

        return devolver;
}
/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxStatusData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxStatusData::descompactar(unsigned char *buffer_global) {

        // Lo primero es poner a 0 los valores de la clase por si habíamos recibido
        // un mensaje anterior
        s1 = 0;
        s2 = 0;

        // Copiamos en las variables de la clase los bytes correspondientes del
buffer_global
        s1 = buffer_global[2];
        s2 = buffer_global[3];

}

/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxVruRateData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxVruRateData::descompactar(unsigned char *buffer_global) {

        // Lo primero es poner a 0 los valores de la clase por si habíamos recibido
        // un mensaje anterior
        s1 = 0;
        s2 = 0;
        azimuth_rate = 0;
        pitch_rate = 0;
        roll_rate = 0;

        // Copiamos en las variables de la clase los bytes correspondientes del
buffer_global
        s1 = buffer_global[2];
        s2 = buffer_global[3];

        // Mediante la función bucle_descompactar colocamos los valores
        azimuth_rate = bucle_descompactar(1,2,buffer_global);
        pitch_rate = bucle_descompactar(3,4,buffer_global);
        roll_rate = bucle_descompactar(5,6,buffer_global);
}
```

```
/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxConfigurationData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxConfigurationData::descompactar(unsigned char *buffer_global) {

	// Lo primero es poner a 0 los valores de la clase por si habíamos recibido
	// un mensaje anterior
	s1 = 0;
	s2 = 0;
	a = 0;
	alfa = 0;
	b = 0;
	beta = 0;
	configuration_flag = 0;
	gamma = 0;
	incremento_x = 0;
	incremento_y = 0;
	incremento_z = 0;
	interval_odometer = 0;
	interval_zupt = 0;
	manual_align_time = 0;
	normal_align_time = 0;
	realign_time = 0;
	stored_heading_align_time = 0;
	tau = 0;
	vms = 0;
	vru_frame_rotation_matrix = 0;


	// Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
	s1 = buffer_global[2];
	s2 = buffer_global[3];

	// Mediante la función bucle_descompactar colocamos los restantes valores
	alfa = bucle_descompactar(1,2,buffer_global);
	beta = bucle_descompactar(3,4,buffer_global);
	gamma = bucle_descompactar(5,6,buffer_global);
	a = bucle_descompactar(7,8,buffer_global);
	b = bucle_descompactar(9,10,buffer_global);
	tau = bucle_descompactar(11,12,buffer_global);
	incremento_x = buffer_global[16];
	incremento_y = buffer_global[17];
	incremento_z = buffer_global[18];
	interval_zupt = buffer_global[19];
	interval_odometer = buffer_global[20];
	normal_align_time = buffer_global[21];
	stored_heading_align_time = buffer_global[22];
	realign_time = buffer_global[23];
	manual_align_time = buffer_global[24];
	configuration_flag = bucle_descompactar(23,24,buffer_global);
	vms = bucle_descompactar(25,26,buffer_global);
	vru_frame_rotation_matrix = configuration_flag =
bucle_descompactar(31,33,buffer_global);

}




/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxAttitudeData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxAttitudeData::descompactar(unsigned char *buffer_global)    {

	// Lo primero es poner a 0 los valores de la clase por si habíamos recibido
	// un mensaje anterior
	s1 = 0;
	s2 = 0;
	pointing_device_azimuth_rate = 0;
```

```
        pointing_device_geodetic = 0;
        pointing_device_pitch = 0;
        pointing_device_pitch_rate = 0;
        vehicle_cant = 0;
        vehicle_pitch = 0;

        // Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
        s1 = buffer_global[2];
        s2 = buffer_global[3];

        // Mediante la función bucle_descompactar colocamos los restantes valores
        pointing_device_geodetic = bucle_descompactar(1,2,buffer_global);
        pointing_device_pitch = bucle_descompactar(3,4,buffer_global);
        vehicle_cant = bucle_descompactar(5,6,buffer_global);
        vehicle_pitch = bucle_descompactar(7,8,buffer_global);
        pointing_device_azimuth_rate = bucle_descompactar(9,10,buffer_global);
        pointing_device_pitch_rate = bucle_descompactar(11,12,buffer_global);
}

/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxBuiltInTest
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxBuiltInTest::descompactar(unsigned char *buffer_global) {

        // Lo primero es poner a 0 los valores de la clase por si habíamos recibido
        // un mensaje anterior
        s1 = 0;
        s2 = 0;
        d1 = 0;
        d2 = 0;

        // Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
        s1 = buffer_global[2];
        s2 = buffer_global[3];
        d1 = buffer_global[4];
        d2 = buffer_global[5];
}



/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxTravelLockData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxTravelLockData::descompactar(unsigned char *buffer_global) {

        // Lo primero es poner a 0 los valores de la clase por si habíamos recibido
        // un mensaje anterior
        s1 = 0;
        s2 = 0;
        pointing_device_geodetic_azimuth = 0;
        pointing_device_pitch = 0;
        travel_lock_pointing_device_geodetic = 0;
        travel_lock_poinging_device_pitch_reference = 0;
        pointing_device_azimuth_rate = 0;
        pointing_device_pitch_rate = 0;

        // Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
        s1 = buffer_global[2];
        s2 = buffer_global[3];

        // Mediante la función bucle_descompactar colocamos los restantes valores
        pointing_device_geodetic_azimuth = bucle_descompactar(1,2,buffer_global);
        pointing_device_pitch = bucle_descompactar(3,4,buffer_global);
        travel_lock_pointing_device_geodetic = bucle_descompactar(5,6,buffer_global);
        travel_lock_poinging_device_pitch_reference =
bucle_descompactar(7,8,buffer_global);
        pointing_device_azimuth_rate = bucle_descompactar(9,10,buffer_global);
        pointing_device_pitch_rate = bucle_descompactar(11,12,buffer_global);
}
```

```
/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxVruOrientation
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxVruOrientation::descompactar(unsigned char *buffer_global) {

	// Lo primero es poner a 0 los valores de la clase por si habíamos recibido
	// un mensaje anterior
	s1 = 0;
	s2 = 0;
	vru_geodetic_azimuth = 0;
	vru_pitch = 0;
	vru_roll = 0;
	pointing_device_geodetic_azimuth = 0;
	pointing_device_pitch = 0;
	pointing_device_cant = 0;

	// Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
	s1 = buffer_global[2];
	s2 = buffer_global[3];

	// Mediante la función bucle_descompactar colocamos los restantes valores
	vru_geodetic_azimuth = bucle_descompactar(1,2,buffer_global);
	vru_pitch = bucle_descompactar(3,4,buffer_global);
	vru_roll = bucle_descompactar(5,6,buffer_global);
	pointing_device_geodetic_azimuth = bucle_descompactar(7,8,buffer_global);
	pointing_device_pitch = bucle_descompactar(9,10,buffer_global);
	pointing_device_cant = bucle_descompactar(11,12,buffer_global);
}

/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxPointingDeviceBoresightAngles
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxPointingDeviceBoresightAngles::descompactar(unsigned char *buffer_global)
{

	// Lo primero es poner a 0 los valores de la clase por si habíamos recibido
	// un mensaje anterior
	s1 = 0;
	s2 = 0;
	a = 0;
	b = 0;
	tau = 0;

	// Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
	s1 = buffer_global[2];
	s2 = buffer_global[3];

	// Mediante la función bucle_descompactar colocamos los restantes valores
	a = bucle_descompactar(1,2,buffer_global);
	b = bucle_descompactar(3,4,buffer_global);
	tau = bucle_descompactar(5,6,buffer_global);
}
/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxExpandedStatusData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxExpandedStatusData::descompactar(unsigned char *buffer_global) {

	// Lo primero es poner a 0 los valores de la clase por si habíamos recibido
	// un mensaje anterior
	s1 = 0;
	s2 = 0;
	d1 = 0;
	d2 = 0;

	// Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
	s1 = buffer_global[2];
	s2 = buffer_global[3];
```

```
        d1 = buffer_global[4];
        d2 = buffer_global[5];
}
/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxSurveyQuality
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxSurveyQuality::descompactar(unsigned char *buffer_global) {

        // Lo primero es poner a 0 los valores de la clase por si habíamos recibido
        // un mensaje anterior
        s1 = 0;
        s2 = 0;
        azimuth_error = 0;
        horizontal_position_error = 0;
        altitude_error = 0;

        // Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
        s1 = buffer_global[2];
        s2 = buffer_global[3];

        // Mediante la función bucle_descompactar colocamos los restantes valores
        azimuth_error = bucle_descompactar(1,2,buffer_global);
        horizontal_position_error = bucle_descompactar(3,4,buffer_global);
        altitude_error = bucle_descompactar(5,6,buffer_global);
}
/************************************************
 - Función: Descompactar
 - Clase: CMensajeRxExtendedConfigurationData
 - Descripción: Esta función se encarga de pasar el array buffer_global
 a los valores de la clase correspondiente
************************************************/
void CMensajeRxExtendedConfigurationData::descompactar(unsigned char *buffer_global) {

        // Lo primero es poner a 0 los valores de la clase por si habíamos recibido
        // un mensaje anterior
        s1 = 0;
        s2 = 0;
        x_vru_to_gps = 0;
        y_vru_to_gps = 0;
        z_vru_to_gps = 0;
        x_vms_to_vru = 0;
        y_vms_to_vru = 0;
        z_vms_to_vru = 0;
        extended_configuration_flags = 0;
        alfa = 0;
        beta = 0;
        gamma = 0;

        // Copiamos en las variables de la clase los bytes correspondientes a S1 y S2
        s1 = buffer_global[2];
        s2 = buffer_global[3];

        // Mediante la función bucle_descompactar colocamos los restantes valores
        x_vru_to_gps = bucle_descompactar(1,2,buffer_global);
        y_vru_to_gps = bucle_descompactar(3,4,buffer_global);
        z_vru_to_gps = bucle_descompactar(5,6,buffer_global);
        x_vms_to_vru = bucle_descompactar(7,8,buffer_global);
        y_vms_to_vru = bucle_descompactar(9,10,buffer_global);
        z_vms_to_vru = bucle_descompactar(11,12,buffer_global);
        extended_configuration_flags = bucle_descompactar(17,18,buffer_global);
        alfa = bucle_descompactar(19,20,buffer_global);
        beta = bucle_descompactar(21,22,buffer_global);
        gamma = bucle_descompactar(23,24,buffer_global);
}
```

Fichero: mensajes.h

```
// mensajes.h: interface for the CCabeceraTx class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MENSAJES_H__5A2FCDA7_D9CE_46EE_9339_43173A18DD1B__INCLUDED_)
```

```
#define AFX_MENSAJES_H__5A2FCDA7_D9CE_46EE_9339_43173A18DD1B__INCLUDED_

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

/*************************************************************
 Constantes asociadas a las clases de los mensajes con la IMU
*************************************************************/
#define TAM_COMPACTADA 50
#define RAD_MILS 1018.59164
#define MASCARA_5_LSB_BITS 31
#define MASCARA_3_MSB_BITS 224
#define READ_TIMEOUT 500 // Milisegundos



/*********************************************************************
 Clase: CCabeceraTx
 Descripción: Cabecera común que tienen todos los mensajes que se transmiten
 Variables:
        - bytes: Indica el número de bytes del paquete
        - secuencia: Número de secuencia
        - codigo: Código del mensaje (Cada mensaje tiene uno propio)
*********************************************************************/
class CCabeceraTx
{
public:
        CCabeceraTx(int b, int s, int c) : bytes(b),secuencia(s),codigo(c) {}
        int bytes;
        int secuencia;
        int codigo;
        unsigned char compactada[TAM_COMPACTADA];

        // Funciones
        void bucle_compactar(int numero, int byte_min, int byte_max);

};


/*********************************************************************
 Clase: CMensajeTxAcceptPosition (13 bytes)
 Herencia: CCabeceraTx
 Variables:
        - northing: 3 bytes
        - altitude: 2 bytes
        - extended_zone: 1 byte
        - hemisphere: 1 byte
        - prime_system_spheroid: 1 byte
        - easting: 3 bytes
*********************************************************************/
class CMensajeTxAcceptPosition : public CCabeceraTx {
public:
        CMensajeTxAcceptPosition() : CCabeceraTx(13,0,5) {}
        int northing;
        int altitude;
        int extended_zone;
        int hemisphere;
        int prime_system_spheroid;
        int easting;

        void compactar();

};


/*********************************************************************
 Clase: CMensajeTxAcceptPointingDeviceBoresight (8 bytes)
 Herencia: CCabeceraTx
 Variables:
        - a: 2 bytes
        - b: 2 bytes
        - gamma: 2 bytes

*********************************************************************/
class CMensajeTxAcceptPointingDeviceBoresight : public CCabeceraTx {
```

```
public:
	CMensajeTxAcceptPointingDeviceBoresight() : CCabeceraTx(8,0,23) {}
	int a;
	int b;
	int gamma;

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxAcceptConfigurationData (35 bytes)
 Herencia: CCabeceraTx
 Variables:
	- alfa: 2 bytes
	- beta: 2 bytes
	- gamma: 2 bytes
	- incremento_x: 1 byte
	- incremento_y: 1 byte
	- incremento_z: 1 byte
	- zupt_zupt: 1 byte
	- zupt_odometer: 1 byte
	- normal_align_time: 1 byte
	- stored_heading_align_time: 1 byte
	- realign_time: 1 byte
	- manual_align_time: 1 byte
	- configuration_flag: 2 bytes
	- vms: 2 bytes
	- vru_matrix: 2 bytes
*********************************************************************/
class CMensajeTxAcceptConfigurationData : public CCabeceraTx {
public:
	CMensajeTxAcceptConfigurationData() : CCabeceraTx(35,0,3) {}
	int alfa;
	int beta;
	int gamma;
	int incremento_x;
	int incremento_y;
	int incremento_z;
	int zupt_zupt;
	int zupt_odometer;
	int normal_align_time;
	int stored_heading_align_time;
	int realign_time;
	int manual_align_time;
	int configuration_flag;
	int vms;
	int vru_matrix;

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnVruRateData (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnVruRateData : public CCabeceraTx {
public:
	CMensajeTxReturnVruRateData() : CCabeceraTx(2,0,13) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnConfigurationData (3 bytes)
 Herencia: CCabeceraTx
 Variables:
	- subcode: 1 byte
*********************************************************************/
class CMensajeTxReturnConfigurationData : public CCabeceraTx {
public:
	CMensajeTxReturnConfigurationData() : CCabeceraTx(3,0,20) {}
	int subcode;

	void compactar();
};
```

```
/*********************************************************************
 Clase: CMensajeTxReturnAttitudeData (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnAttitudeData : public CCabeceraTx {
public:
	CMensajeTxReturnAttitudeData() : CCabeceraTx(2,0,1) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnBuiltInTest (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnBuiltInTest : public CCabeceraTx {
public:
	CMensajeTxReturnBuiltInTest() : CCabeceraTx(2,0,31) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnTravelLockData (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnTravelLockData : public CCabeceraTx {
public:
	CMensajeTxReturnTravelLockData() : CCabeceraTx(2,0,0) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnVruOrientation (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnVruOrientation : public CCabeceraTx {
public:
	CMensajeTxReturnVruOrientation() : CCabeceraTx(2,0,12) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnStatusData (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnStatusData : public CCabeceraTx {
public:
	CMensajeTxReturnStatusData() : CCabeceraTx(2,0,16) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnPointingDeviceBoresightAngles (3 bytes)
 Herencia: CCabeceraTx
 Variables:
	- subcode: 1 byte
*********************************************************************/
class CMensajeTxReturnPointingDeviceBoresightAngles : public CCabeceraTx {
public:
	CMensajeTxReturnPointingDeviceBoresightAngles() : CCabeceraTx(3,0,20) {}
	int subcode;
	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnExpandedStatusData (3 bytes)
```

```
 Herencia: CCabeceraTx
 Variables:
       - subcode: 1 byte
*********************************************************************/
class CMensajeTxReturnExpandedStatusData : public CCabeceraTx {
public:
       CMensajeTxReturnExpandedStatusData() : CCabeceraTx(3,0,20) {}
       int subcode;
       void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnSurveyQuality (3 bytes)
 Herencia: CCabeceraTx
 Variables:
       - subcode: 1 byte
*********************************************************************/
class CMensajeTxReturnSurveyQuality : public CCabeceraTx {
public:
       CMensajeTxReturnSurveyQuality() : CCabeceraTx(3,0,20) {}
       int subcode;
       void compactar();
};

/*********************************************************************
 Clase: CMensajeTxOverrideAlert (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxOverrideAlert : public CCabeceraTx {
public:
       CMensajeTxOverrideAlert() : CCabeceraTx(2,0,8) {}

       void compactar();
};

/*********************************************************************
 Clase: CMensajeTxReturnExtendedConfigurationData (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxReturnExtendedConfigurationData : public CCabeceraTx {
public:
       CMensajeTxReturnExtendedConfigurationData() : CCabeceraTx(4,0,6) {}
       int subcode;

       void compactar();
};

/*********************************************************************
 Clase: CMensajeTxResetDistance (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxResetDistance : public CCabeceraTx {
public:
       CMensajeTxResetDistance() : CCabeceraTx(2,0,7) {}

       void compactar();
};

/*********************************************************************
 Clase: CMensajeTxRestart (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxRestart : public CCabeceraTx {
public:
       CMensajeTxRestart() : CCabeceraTx(2,0,26) {}

       void compactar();
};

/*********************************************************************
 Clase: CMensajeTxShutdown (2 bytes)
 Herencia: CCabeceraTx
```

```
 Variables: Ninguna
*********************************************************************/
class CMensajeTxShutdown : public CCabeceraTx {
public:
	CMensajeTxShutdown() : CCabeceraTx(2,0,20) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxEnableZeroVelocityUpdates (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxEnableZeroVelocityUpdates : public CCabeceraTx {
public:
	CMensajeTxEnableZeroVelocityUpdates() : CCabeceraTx(2,0,21) {}

	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxInTravelLock (3 bytes)
 Herencia: CCabeceraTx
 Variables:
	- subcode
*********************************************************************/
class CMensajeTxInTravelLock : public CCabeceraTx {
public:
	CMensajeTxInTravelLock() : CCabeceraTx(3,0,20) {}
	int subcode;
	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxOdometerModeRequest (3 bytes)
 Herencia: CCabeceraTx
 Variables:
	- subcode
*********************************************************************/
class CMensajeTxOdometerModeRequest : public CCabeceraTx {
public:
	CMensajeTxOdometerModeRequest() : CCabeceraTx(3,0,20) {}
	int subcode;
	void compactar();
};

/*********************************************************************
 Clase: CMensajeTxRealign (2 bytes)
 Herencia: CCabeceraTx
 Variables: Ninguna
*********************************************************************/
class CMensajeTxRealign : public CCabeceraTx {
public:
	CMensajeTxRealign() : CCabeceraTx(2,0,28) {}

	void compactar();
};


/*********************************
 Clase: CCabeceraRx_recepcion
 Descripción: Cabecera común que tienen todos los mensajes que se reciben
 Variables:
	- bytes: Indica el número de bytes del paquete
	- secuencia: Número de secuencia
	- identificador: Código del mensaje (Cada mensaje tiene uno propio)
*********************************/
class CCabeceraRx{
public:
	CCabeceraRx(int b, int s, int i) : bytes(b),secuencia(s),identificador(i) {}
	char *buffer_recepcion;
	int bytes;
	int secuencia;
	int identificador;
```

```
        virtual void descompactar(unsigned char *buffer_global);
        int bucle_descompactar(int byte_min, int byte_max, unsigned char *buffer);
};

/*****************************************************************
 Clase: CMensajeRxStatusData (2 bytes)
 Herencia: CCabeceraRx
 Variables: Ninguna
*****************************************************************/
class CMensajeRxStatusData : public CCabeceraRx {
public:
        CMensajeRxStatusData() : CCabeceraRx(4,0,0) {}
        int s1;
        int s2;

        void descompactar(unsigned char *buffer_global);
};

/*****************************************************************
 Clase: CMensajeRxVruRateData (10 bytes)
 Herencia: CCabeceraRx
 Variables:
        - Azimuth Rate
        - Pitch Rate
        - Roll Rate
*****************************************************************/
class CMensajeRxVruRateData : public CCabeceraRx {
public:
        CMensajeRxVruRateData() : CCabeceraRx(10,0,11) {}
        int s1;
        int s2;
        int azimuth_rate;
        int pitch_rate;
        int roll_rate;

        void descompactar(unsigned char *buffer_global);
};


/*****************************************************************
 Clase: CMensajeRxConfigurationData (37 bytes)
 Herencia: CCabeceraRx
 Variables:
        - alfa
        - beta
        - gamma
        - a
        - b
        - tau
        - incremento_x
        - incremento_y
        - incremento_z
        - interval_zupt
        - interval_odometer
        - normal_align_time
        - stored_heading_align_time
        - realign_time
        - manual_align_time
        - configuration_flag
        - vms
        - vru_frame_rotation_matrix
*****************************************************************/
class CMensajeRxConfigurationData : public CCabeceraRx {
public:
        CMensajeRxConfigurationData() : CCabeceraRx(37,0,18) {}

        int s1;
        int s2;
        int alfa;
        int beta;
        int gamma;
        int a;
        int b;
        int tau;
        int incremento_x;
        int incremento_y;
```

```
        int incremento_z;
        int interval_zupt;
        int interval_odometer;
        int normal_align_time;
        int stored_heading_align_time;
        int realign_time;
        int manual_align_time;
        int configuration_flag;
        int vms;
        int vru_frame_rotation_matrix;

        void descompactar(unsigned char *buffer_global);
};

/*********************************************************************
 Clase: CMensajeRxAttitudeData (16 bytes)
 Herencia: CCabeceraRx
 Variables:
        - pointing_device_geodetic
        - pointing_device_pitch
        - vehicle_cant
        - vehicle_pitch
        - pointing_device_azimuth_rate
        - pointing_device_pitch_rate
*********************************************************************/
class CMensajeRxAttitudeData : public CCabeceraRx {
public:
        CMensajeRxAttitudeData() : CCabeceraRx(16,0,2) {}

        int s1;
        int s2;
        int pointing_device_geodetic;
        int pointing_device_pitch;
        int vehicle_cant;
        int vehicle_pitch;
        int pointing_device_azimuth_rate;
        int pointing_device_pitch_rate;

        void descompactar(unsigned char *buffer_global);
};


/*********************************************************************
 Clase: CMensajeRxBuiltInTest (16 bytes)
 Herencia: CCabeceraRx
 Variables:
        - d1
        - d2
*********************************************************************/
class CMensajeRxBuiltInTest : public CCabeceraRx {
public:
        CMensajeRxBuiltInTest() : CCabeceraRx(6,0,4) {}

        int s1;
        int s2;
        int d1;
        int d2;

        void descompactar(unsigned char *buffer_global);
};




/*********************************************************************
 Clase: CMensajeRxTravelLockData (16 bytes)
 Herencia: CCabeceraRx
 Variables:
        - pointing_device_geodetic_azimuth
        - pointing_device_pitch
        - travel_lock_pointing_device_geodetic
        - travel_lock_poinging_device_pitch_reference
        - pointing_device_azimuth_rate
        - pointing_device_pitch_rate
*********************************************************************/
class CMensajeRxTravelLockData : public CCabeceraRx {
```

```
public:
	CMensajeRxTravelLockData() : CCabeceraRx(16,0,3) {}

	int s1;
	int s2;
	int pointing_device_geodetic_azimuth;
	int pointing_device_pitch;
	int travel_lock_pointing_device_geodetic;
	int travel_lock_poinging_device_pitch_reference;
	int pointing_device_azimuth_rate;
	int pointing_device_pitch_rate;

	void descompactar(unsigned char *buffer_global);
};

/*********************************************************************
 Clase: CMensajeRxVruOrientation (16 bytes)
 Herencia: CCabeceraRx
 Variables:
	- vru_geodetic_azimuth
	- vru_pitch
	- vru_roll
	- pointing_device_geodetic_azimuth
	- pointing_device_pitch
	- pointing_device_cant
*********************************************************************/
class CMensajeRxVruOrientation : public CCabeceraRx {
public:
	CMensajeRxVruOrientation() : CCabeceraRx(16,0,10) {}

	int s1;
	int s2;
	int vru_geodetic_azimuth;
	int vru_pitch;
	int vru_roll;
	int pointing_device_geodetic_azimuth;
	int pointing_device_pitch;
	int pointing_device_cant;

	void descompactar(unsigned char *buffer_global);
};

/*********************************************************************
 Clase: CMensajeRxPointingDeviceBoresightAngles (10 bytes)
 Herencia: CCabeceraRx
 Variables:
	- a
	- b
	- tau
*********************************************************************/
class CMensajeRxPointingDeviceBoresightAngles : public CCabeceraRx {
public:
	CMensajeRxPointingDeviceBoresightAngles() : CCabeceraRx(10,0,20) {}

	int s1;
	int s2;
	int a;
	int b;
	int tau;

	void descompactar(unsigned char *buffer_global);
};
/*********************************************************************
 Clase: CMensajeRxExpandedStatusData (6 bytes)
 Herencia: CCabeceraRx
 Variables:
	- d1
	- d2
*********************************************************************/
class CMensajeRxExpandedStatusData : public CCabeceraRx {
public:
	CMensajeRxExpandedStatusData() : CCabeceraRx(6,0,31) {}

	int s1;
	int s2;
	int d1;
```

```
        int d2;

        void descompactar(unsigned char *buffer_global);
};
/*********************************************************************
 Clase: CMensajeRxSurveyQuality (10 bytes)
 Herencia: CCabeceraRx
 Variables:
        - azimuth_error
        - horizontal_position_error
        - altitude_error
*********************************************************************/
class CMensajeRxSurveyQuality : public CCabeceraRx {
public:
        CMensajeRxSurveyQuality() : CCabeceraRx(10,0,21) {}

        int s1;
        int s2;
        int azimuth_error;
        int horizontal_position_error;
        int altitude_error;

        void descompactar(unsigned char *buffer_global);
};
/*********************************************************************
 Clase: CMensajeRxExtendedConfigurationData (34 bytes)
 Herencia: CCabeceraRx
 Variables:
        - azimuth_error
        - horizontal_position_error
        - altitude_error
*********************************************************************/
class CMensajeRxExtendedConfigurationData : public CCabeceraRx {
public:
        CMensajeRxExtendedConfigurationData() : CCabeceraRx(10,0,21) {}

        int s1;
        int s2;
        int x_vru_to_gps;
        int y_vru_to_gps;
        int z_vru_to_gps;
        int x_vms_to_vru;
        int y_vms_to_vru;
        int z_vms_to_vru;
        int extended_configuration_flags;
        int alfa;
        int beta;
        int gamma;

        void descompactar(unsigned char *buffer_global);
};

#endif // !defined(AFX_MENSAJES_H__5A2FCDA7_D9CE_46EE_9339_43173A18DD1B__INCLUDED_)
```

Fichero: SerialCom.cpp

```
// SerialCom.cpp : implementation file
//

#include "stdafx.h"
#include "SerialCom.h"

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


//////////////////////////////////////////////////////////////////////
// SerialCom.cpp: implementación de la clase SerialCom

// CSerialCom

CSerialCom::CSerialCom()
```

```
{
}

CSerialCom::~CSerialCom()
{
}


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


/////////////////////////////////////////////////////////////////////////////
// CSerialCom message handlers

BOOL CSerialCom::OpenPort(CString portname)
{
//portname= "//./" +portname;

hComm = CreateFile(portname,
			GENERIC_READ | GENERIC_WRITE,
			0,
			0,
			OPEN_EXISTING,
			0,
			0);
		if(hComm==INVALID_HANDLE_VALUE){
			AfxMessageBox(_T("Error abriendo el puerto"));
		return false;}
		else
			return true;

}

BOOL CSerialCom::ConfigurePort(DWORD BaudRate, BYTE ByteSize, DWORD fParity, BYTE
Parity, BYTE StopBits)
{
	if((m_bPortReady = GetCommState(hComm, &m_dcb))==0){
		AfxMessageBox(_T("Error de GetCommState"));

		CloseHandle(hComm);
	return false;}
m_dcb.BaudRate =BaudRate;
m_dcb.ByteSize = ByteSize;
m_dcb.Parity =Parity ;
m_dcb.StopBits =StopBits;
m_dcb.fBinary=TRUE;
m_dcb.fDsrSensitivity=false;
m_dcb.fParity=fParity;
m_dcb.fOutX=false;
m_dcb.fInX=false;
m_dcb.fNull=false;
m_dcb.fAbortOnError=TRUE;
m_dcb.fOutxCtsFlow=FALSE;
m_dcb.fOutxDsrFlow=false;
m_dcb.fDtrControl=DTR_CONTROL_DISABLE;
m_dcb.fDsrSensitivity=false;
m_dcb.fRtsControl=RTS_CONTROL_DISABLE;
m_dcb.fOutxCtsFlow=false;
m_dcb.fOutxCtsFlow=false;

m_bPortReady = SetCommState(hComm, &m_dcb);
if(m_bPortReady ==0){
		AfxMessageBox(_T("Error de SetCommState"));
		//MessageBox("SetCommState Error","Error",MB_OK+MB_ICONERROR);
		CloseHandle(hComm);
	return false;}
return true;
}

BOOL CSerialCom::SetCommunicationTimeouts(DWORD ReadIntervalTimeout, DWORD
ReadTotalTimeoutMultiplier, DWORD ReadTotalTimeoutConstant, DWORD
WriteTotalTimeoutMultiplier, DWORD WriteTotalTimeoutConstant)
```

```
{
if((m_bPortReady = GetCommTimeouts (hComm, &m_CommTimeouts))==0)
   return false;
m_CommTimeouts.ReadIntervalTimeout =ReadIntervalTimeout;
m_CommTimeouts.ReadTotalTimeoutConstant =ReadTotalTimeoutConstant;
m_CommTimeouts.ReadTotalTimeoutMultiplier =ReadTotalTimeoutMultiplier;
m_CommTimeouts.WriteTotalTimeoutConstant = WriteTotalTimeoutConstant;
m_CommTimeouts.WriteTotalTimeoutMultiplier =WriteTotalTimeoutMultiplier;
               m_bPortReady = SetCommTimeouts (hComm, &m_CommTimeouts);
               if(m_bPortReady ==0){
                       AfxMessageBox(_T("Error de StCommTimeouts"));
                       //MessageBox("StCommTimeouts function failed","Com Port
Error",MB_OK+MB_ICONERROR);
               CloseHandle(hComm);
               return false;}
               return true;
}

BOOL CSerialCom::WriteByte(unsigned char *compactada)
{

       BOOL fRes = TRUE;
       int i=0;

       // El primer byte de la tabla indica la longitud
       while (i<compactada[0])
       {
               // Aunque WriteFile devuelva un valor no significa que haya terminado
               // de enviar datos por el puerto serie. Luego se comprueba mediante
               // WaitForSingleObject
               if (!WriteFile(hComm, &compactada[i], 1, &iBytesWritten, NULL))
               {
                       // HA HABIDO UN ERROR. ACTIVAMOS LA ALARMA
                       fRes = FALSE;
               }
               else
                       // La operación se completó inmediatamente
                       fRes = TRUE;

               i++;
       }

       return fRes;


       /* ANTIGUO CÓDIGO PARA ENVIAR UN BYTE SIMPLEMENTE */
       /************************************************
       if(WriteFile(hComm,&bybyte,1,&iBytesWritten,NULL)==0)
               return false;
       else
               return true;
       ************************************************/
}

BOOL CSerialCom::ReadByte(BYTE       &resp)
{
BYTE rx;
resp=0;

DWORD dwBytesTransferred=0;

if (ReadFile (hComm, &rx, 1, &dwBytesTransferred, 0)){
                        if (dwBytesTransferred == 1){
                               resp=rx;
                               return true;}}

       return false;
}


void CSerialCom::ClosePort()
{
       CloseHandle(hComm);
       return;
}
```

```
BOOL CSerialCom::lee(char *bufferIn)
{
	int i=0;
	BYTE bufferByte;

	// Hacemos una primera lectura
	if ( ReadByte(bufferByte) )
	{
		// Copiamos ese primer byte
		bufferIn[i] = bufferByte;
		i++;

		while ( ReadByte(bufferByte) )
		{
			bufferIn[i] = bufferByte;
			i++;
		}
		return TRUE;
	}
	else
		return FALSE;

}

// Con esta función sacamos de bufferIn la posible trama que haya y la
// copiamos en bufferOut
bool CSerialCom::Extract(char *bufferIn, char *bufferOut)
{
	//
	return TRUE;
}
```

Fichero: SerialCom.h

```
#if !defined(AFX_SerialCom_H__0DCE7CC1_2426_4BDF_9AFC_410B32D9FE74__INCLUDED_)
#define AFX_SerialCom_H__0DCE7CC1_2426_4BDF_9AFC_410B32D9FE74__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SerialCom.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSerialCom
/////////////////////////////////////////////////////////////////////////////
// SerialCom.h: Implementación de la clase CSerialCom.

class CSerialCom : public CWnd
{
// Constructor
public:
	CSerialCom();

// Operaciones
public:

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

// Implementación
public:
	bool Extract(char *bufferIn, char *bufferOut);
	BOOL lee(char *bufferIn);
	void ClosePort();
	BOOL ReadByte(BYTE &resp);
	BOOL WriteByte(unsigned char *compactada);
	BOOL OpenPort(CString portname);
	BOOL SetCommunicationTimeouts(DWORD ReadIntervalTimeout,DWORD
ReadTotalTimeoutMultiplier,DWORD ReadTotalTimeoutConstant,DWORD
WriteTotalTimeoutMultiplier,DWORD WriteTotalTimeoutConstant);
	BOOL ConfigurePort(DWORD BaudRate,BYTE ByteSize,DWORD fParity,BYTE  Parity,BYTE
StopBits);
```

```
        HANDLE hComm;
        DCB      m_dcb;
        COMMTIMEOUTS m_CommTimeouts;
        BOOL     m_bPortReady;
        BOOL     bWriteRC;
        BOOL     bReadRC;
        DWORD iBytesWritten;
        DWORD iBytesRead;
        DWORD dwBytesRead;
        virtual ~CSerialCom();

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

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

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

#endif // !defined(AFX_SerialCom_H__0DCE7CC1_2426_4BDF_9AFC_410B32D9FE74__INCLUDED_)
```

ANEXO C: CÓDIGO FUENTE DEL MONITOR PC

A continuación se muestra el código fuente del fichero "FicherosDlg.cpp" correspondiente a la aplicación del servidor. En el cdrom adjunto se encuentran ambos proyectos (viper y servidor) con todos los ficheros necesarios para la compilación del software.

Fichero: FicherosDlg.cpp

```
// FicherosDlg.cpp: archivo de implementación
//

#include "stdafx.h"
#include "Ficheros.h"
#include "FicherosDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// Cuadro de diálogo de CFicherosDlg
#define PUERTO_TCP_RECIBIR 6969
#define PUERTO_TCP_ENVIAR 7070

#define RECV_BUFFER_SIZE 4096
#define SEND_BUFFER_SIZE 4096


CFicherosDlg::CFicherosDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CFicherosDlg::IDD, pParent)
	, strIpMovil(_T(""))
	, strIpreferencia(_T(""))
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CFicherosDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_EDIT1, strIpMovil);
	DDX_Text(pDX, IDC_EDIT2, strIpreferencia);
}

BEGIN_MESSAGE_MAP(CFicherosDlg, CDialog)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_BN_CLICKED(IDOK, &CFicherosDlg::OnBnClickedOk)
	ON_BN_CLICKED(IDCANCEL, &CFicherosDlg::OnBnClickedCancel)
	ON_BN_CLICKED(IDC_BUTTON3, &CFicherosDlg::OnBnClickedButton3)
	ON_BN_CLICKED(IDC_BUTTON1, &CFicherosDlg::OnBnClickedButton1)
	ON_BN_CLICKED(IDC_BUTTON4, &CFicherosDlg::OnBnClickedButton4)
	ON_BN_CLICKED(IDC_BUTTON2, &CFicherosDlg::OnBnClickedButton2)
END_MESSAGE_MAP()


// Controladores de mensaje de CFicherosDlg

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

	// Establecer el icono para este cuadro de diálogo. El marco de trabajo realiza
esta operación
	//  automáticamente cuando la ventana principal de la aplicación no es un cuadro
de diálogo
	SetIcon(m_hIcon, TRUE);			// Establecer icono grande
```

```
	SetIcon(m_hIcon, FALSE);		// Establecer icono pequeño

	// TODO: agregar aquí inicialización adicional

	return TRUE;  // Devuelve TRUE  a menos que establezca el foco en un control
}

// Si agrega un botón Minimizar al cuadro de diálogo, necesitará el siguiente código
//  para dibujar el icono. Para aplicaciones MFC que utilicen el modelo de documentos y
vistas,
//  esta operación la realiza automáticamente el marco de trabajo.

void CFicherosDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // Contexto de dispositivo para dibujo

		SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()),
0);

		// Centrar icono en el rectángulo de cliente
		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;

		// Dibujar el icono
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// El sistema llama a esta función para obtener el cursor que se muestra mientras el
usuario arrastra
//  la ventana minimizada.
HCURSOR CFicherosDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}


void CFicherosDlg::OnBnClickedOk()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	OnOK();
}

void CFicherosDlg::OnBnClickedCancel()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	OnCancel();
}

bool CFicherosDlg::recibe_fichero(CString nombre_fichero, CString ip)
{
	// Creamos el socket y nos conectamos
	CSocket sockClient;
	sockClient.Create();
	sockClient.Connect(ip, PUERTO_TCP_RECIBIR);

	// Variable de retorno
	BOOL bRet = TRUE;

	// Variables utilizadas en la operación de recepción
	int dataLength, cbBytesRet, cbLeftToReceive;

	// Puntero al buffer de recepción
	BYTE* recdData = NULL;

	// Fichero donde se almacenan los datos recibidos
```

```
	CFile destFile;
	CFileException fe;
	BOOL bFileIsOpen = FALSE;

	// Abrimos el fichero de recepción
	if( !( bFileIsOpen = destFile.Open( nombre_fichero, CFile::modeCreate |
		   CFile::modeWrite | CFile::typeBinary, &fe ) ) )
	{
		TCHAR strCause[256];
		fe.GetErrorMessage( strCause, 255 );
		AfxMessageBox("ERROR CON EL NOMBRE DEL FICHERO");
		/* Manejamos el error */

		bRet = FALSE;
		// Saltamos al final de la función
		goto PreReturnCleanup;
	}

	// Lo primero que recibimos es el tamaño del fichero
	cbLeftToReceive = sizeof( dataLength );

	// Bucle de recepción de bytes mientras haya bytes que recibir
	do
	{
		BYTE* bp = (BYTE*)(&dataLength) + sizeof(dataLength) - cbLeftToReceive;
		cbBytesRet = sockClient.Receive( bp, cbLeftToReceive );

		// Comprobamos errores
		if ( cbBytesRet == SOCKET_ERROR || cbBytesRet == 0 )
		{
			int iErr = ::GetLastError();
			AfxMessageBox("ERROR DE RECEPCION DE LONGITUD");
			/* Manejamos el error */

			bRet = FALSE;
			goto PreReturnCleanup;
		}

		// Decrementamos los bytes a recibir
		cbLeftToReceive -= cbBytesRet;
	}
	while ( cbLeftToReceive > 0 );

	dataLength = ntohl( dataLength );

	// Ahora recibimos tramas del fichero y la almacenamos en el buffer
	recdData = new byte[RECV_BUFFER_SIZE];
	cbLeftToReceive = dataLength;

	// Nuevo bucle mientras haya datos que recibir
	do
	{
		int iiGet, iiRecd;

		iiGet = (cbLeftToReceive<RECV_BUFFER_SIZE) ?
					cbLeftToReceive : RECV_BUFFER_SIZE ;
		iiRecd = sockClient.Receive( recdData, iiGet );

		// Comprobamos errores
		if ( iiRecd == SOCKET_ERROR || iiRecd == 0 )
		{
			int iErr = ::GetLastError();
			AfxMessageBox("ERROR A LA HORA DE RECIBIR EL FICHERO");
			/* Manejamos el error */

			bRet = FALSE;
			goto PreReturnCleanup;
		}

		destFile.Write( recdData, iiRecd); // Write it
		cbLeftToReceive -= iiRecd;

	}
	while ( cbLeftToReceive > 0 );

	PreReturnCleanup: // labelled "goto" destination
```

```
	// Borramos la memoria utilizado, cerramos socket y fichero
	delete[] recdData;

	if ( bFileIsOpen )
		destFile.Close();

	sockClient.Close();

	return bRet;
}

void CFicherosDlg::OnBnClickedButton3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	// Leemos la IP de la interfaz
	UpdateData(TRUE);

	// Comprobamos que haya introducido la IP
	if (strIpMovil != "")
	{
		// Preguntamos por el nombre del fichero con el que quieren salvar los
datos
		CString nombre_fichero;

		// Constructor del cuadro de diálogo
		CFileDialog fichero(FALSE, "csv", "resultado.csv", NULL,"Ficheros CSV
(*.csv)|*.csv||", this);
		fichero.m_pOFN->lpstrTitle="Fichero de resultados";
		fichero.m_pOFN->lpstrInitialDir="c:";


		// Si el usuario pulsa sobre el botón guardar en el cuadro de diálogo
		if ( fichero.DoModal() == IDOK )
		{
			// Guardamos el nombre del fichero junto a la ruta
			nombre_fichero = fichero.GetPathName();

			// LLamamos a la función encargada de recibir los datos
			if (recibe_fichero(nombre_fichero,strIpMovil))
				AfxMessageBox("Fichero recibido correctamente");
			else
				AfxMessageBox("No se ha podido recibir el fichero");
		}
	}
	else
		AfxMessageBox("No ha introducido la IP");



}

void CFicherosDlg::OnBnClickedButton1()
{
	// Leemos la IP de la interfaz
	UpdateData(TRUE);

	// Comprobamos que haya introducido la IP
	if (strIpMovil != "")
	{
		// Preguntamos por el nombre del fichero con el que quieren salvar los
datos
		CString nombre_fichero;

		// Constructor del cuadro de diálogo
		CFileDialog fichero(TRUE, "txt", "conf.txt", NULL,"Ficheros TXT
(*.txt)|*.txt||", this);
		fichero.m_pOFN->lpstrTitle="Selecciona el fichero de configuración";
		fichero.m_pOFN->lpstrInitialDir="c:";


		// Si el usuario pulsa sobre el botón guardar en el cuadro de diálogo
		if ( fichero.DoModal() == IDOK )
		{
			// Guardamos el nombre del fichero junto a la ruta
			nombre_fichero = fichero.GetPathName();
```

```
                        CWnd::SetWindowText(_T("ENVIANDO CONFIGURACIÓN A LA VIPER"));
                        // LLamamos a la función encargada de enviar el fichero
                        if (envia_fichero(nombre_fichero,strIpMovil))
                                AfxMessageBox("Fichero enviado correctamente");
                        else
                                AfxMessageBox("No se ha podido enviar el fichero");
                        CWnd::SetWindowText(_T("SERVIDOR PC"));
                }
        }
        else
                AfxMessageBox("No ha introducido la IP");
}

bool CFicherosDlg::envia_fichero(CString nombre_fichero, CString ip)
{
        // Creamos el socket
        CSocket sockSrvr;
        sockSrvr.Create(PUERTO_TCP_ENVIAR);
        sockSrvr.Listen();

        // Usamos otro CSocket para aceptar la conexión
        CSocket sockConnection;
        sockSrvr.Accept(sockConnection);

        // Variables locales utilizas en la transferencia del fichero

        // Indica si la operación se ha producido correctamente o no
        BOOL bRet = TRUE;
        // Longitud del fichero y número de bytes restantes por enviar
        int fileLength, cbLeftToSend;
        // Puntero hacia el buffer de memoria
        // La memoria se reserva tras enviar  el tamaño del fichero
        BYTE* sendData = NULL;

        CFile sourceFile;
        CFileException fe;
        BOOL bFileIsOpen = FALSE;

        if( !( bFileIsOpen = sourceFile.Open( nombre_fichero,
                   CFile::modeRead | CFile::typeBinary, &fe ) ) )
        {
                TCHAR strCause[256];
                fe.GetErrorMessage( strCause, 255 );
                AfxMessageBox(strCause);
                /* Si hay algún error en la apertura del fichero lo manejamos aquí */

                bRet = FALSE;
                goto PreReturnCleanup;
        }

        // Primero enviamos la longitudo del fichero
        fileLength = sourceFile.GetLength();
        fileLength = htonl( fileLength );

        // El número de bytes a enviar es el tamaño de un entero
        cbLeftToSend = sizeof( fileLength );

        // Bucle para enviar la longitud del fichero
        do
        {
                int cbBytesSent;
                BYTE* bp = (BYTE*)(&fileLength) + sizeof(fileLength) - cbLeftToSend;
                cbBytesSent = sockConnection.Send( bp, cbLeftToSend );

                // Si ha habido algún error lo tratamos y salimos del bucle
                if ( cbBytesSent == SOCKET_ERROR )
                {
                        int iErr = ::GetLastError();
                        AfxMessageBox(_T("ERROR 2"));
                        bRet = FALSE;
                        goto PreReturnCleanup;
                }

                // Los datos se han enviado correctamente, por lo que se resta
                // de los bytes por enviar
```

```
                cbLeftToSend -= cbBytesSent;
        }
        while ( cbLeftToSend>0 );


        // Ahora enviamos los datos del fichero
        sendData = new BYTE[SEND_BUFFER_SIZE];

        cbLeftToSend = sourceFile.GetLength();

        // Bucle de envío
        do
        {
                // Leemos un conjunto de SEND_BUFFER_SIZE bytes desde el fichero
                int sendThisTime, doneSoFar, buffOffset;

                sendThisTime = sourceFile.Read( sendData, SEND_BUFFER_SIZE );
                buffOffset = 0;

                do
                {
                        doneSoFar = sockConnection.Send( sendData + buffOffset,

                sendThisTime );
                        // Comprobamos si hay error
                        if ( doneSoFar == SOCKET_ERROR )
                        {
                                int iErr = ::GetLastError();
                                bRet = FALSE;
                                goto PreReturnCleanup;
                        }

                        // Dato enviado satisfactoriamente
                        buffOffset += doneSoFar;
                        sendThisTime -= doneSoFar;
                        cbLeftToSend -= doneSoFar;
                }
                while ( sendThisTime > 0 );

        }
        while ( cbLeftToSend > 0 );

        // Etiqueta
        PreReturnCleanup:

        // Liberamos la memoria utilizada
        delete[] sendData;

        // Cerramos el descriptor si se ha abierto
        if ( bFileIsOpen )
                sourceFile.Close();

        // Cerramos el socket
        sockConnection.Close();

        return bRet;
}

void CFicherosDlg::OnBnClickedButton4()
{
        // TODO: Agregue aquí su código de controlador de notificación de control
        // Leemos la IP de la interfaz
        UpdateData(TRUE);

        // Comprobamos que haya introducido la IP
        if (strIpreferencia != "")
        {
                // Preguntamos por el nombre del fichero con el que quieren salvar los
datos
                CString nombre_fichero;

                // Constructor del cuadro de diálogo
                CFileDialog fichero(TRUE, "txt", "conf.txt", NULL,"Ficheros TXT
(*.txt)|*.txt||", this);
                fichero.m_pOFN->lpstrTitle="Selecciona el fichero de configuración";
                fichero.m_pOFN->lpstrInitialDir="c:";
```

```
		// Si el usuario pulsa sobre el botón guardar en el cuadro de diálogo
		if ( fichero.DoModal() == IDOK )
		{
			// Guardamos el nombre del fichero junto a la ruta
			nombre_fichero = fichero.GetPathName();

			CWnd::SetWindowText(_T("ENVIANDO CONFIGURACIÓN A LA VIPER"));
			// LLamamos a la función encargada de enviar el fichero
			if (envia_fichero(nombre_fichero,strIpreferencia))
				AfxMessageBox("Fichero enviado correctamente");
			else
				AfxMessageBox("No se ha podido enviar el fichero");
			CWnd::SetWindowText(_T("SERVIDOR PC"));
		}
	}
	else
		AfxMessageBox("No ha introducido la IP");

}

void CFicherosDlg::OnBnClickedButton2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	// Leemos la IP de la interfaz
	UpdateData(TRUE);

	// Comprobamos que haya introducido la IP
	if (strIpMovil != "")
	{
		// Preguntamos por el nombre del fichero con el que quieren salvar los
datos
		CString nombre_fichero;

		// Constructor del cuadro de diálogo
		CFileDialog fichero(TRUE, "txt", "conf.txt", NULL,"Ficheros TXT
(*.txt)|*.txt||", this);
		fichero.m_pOFN->lpstrTitle="Selecciona el fichero de experimentos";
		fichero.m_pOFN->lpstrInitialDir="c:";

		// Si el usuario pulsa sobre el botón guardar en el cuadro de diálogo
		if ( fichero.DoModal() == IDOK )
		{
			// Guardamos el nombre del fichero junto a la ruta
			nombre_fichero = fichero.GetPathName();

			CWnd::SetWindowText(_T("ENVIANDO CONFIGURACIÓN A LA VIPER"));
			// LLamamos a la función encargada de enviar el fichero
			if (envia_fichero(nombre_fichero,strIpMovil))
				AfxMessageBox("Fichero enviado correctamente");
			else
				AfxMessageBox("No se ha podido enviar el fichero");
			CWnd::SetWindowText(_T("SERVIDOR PC"));
		}
	}
	else
		AfxMessageBox("No ha introducido la IP");

}
```