6 ANEXO II: IMPLEMENTACIÓN DEL CÓDIGO DE LA DLL

6.1 Las comunicaciónes: La clase UCom

Las comunicaciones series establecidas para el control del sistema se manejan a través de la clase UCom. La clase tiene funciones para implementar el nivel de enlace del conjunto, en este caso puerto serie RS-422.

Las más destacables son funciones para mandar y recibir cadenas, obtener régimen binario, comprobaciones de asentimientos (Ack, del inglés aknowledge), configuraciones… La clase se define de la siguiente manera:

```
class UCom
{
public:
        UCom();
        virtual ~UCom();

public:
        DWORD BaudRate;
        void CloseCom();
        float GetBytePerSec(void);
        void flash();
        char * PrintNackErr(UINT uErr);
        int NackErrorMessage(UINT uErr);
        BOOL CheckForAck();
        BOOL ReceiveString(char *lpBuffer, DWORD nNumberOfBytesToRead, BOOL
bCheckForAck=false);
        BOOL SendString(char *lpBuffer);
        DWORD ReadTout(char *lpBuffer,DWORD nNumberOfBytesToRead,DWORD
nTimeOutMs,char cEofChar);
        DWORD WriteSetup(CEdit &theCEdit);
        DWORD ReadSetup(CEdit &theCEdit);
        DWORD Read2CEdit(CEdit& theCEdit);
        DWORD Read(char *lpBuffer,DWORD nNumberOfBytesToRead);
        BOOL WriteCh(UINT nChar,CEdit &theCEdit);
        BOOL Write(char *lpBuffer,int len=0);
        void ErrorMessage();
        BOOL Connection();
        HANDLE hCommDev;
        BOOL bDebugMode;

private:
        BOOL bCheckCrcMode;
```

```
	BOOL bCheckLineMode;
	BOOL bCheckReadAll;
	char pCombo_Link[21];
	BOOL bOk;
	COMMCONFIG CC;
	COMMTIMEOUTS CT;
	CSocket *pCSocket;
	HANDLE  hRunMutex;		/* "Keep Running" mutex */
	char pIPAddress[21];
	UINT uPort;
	UINT uLastNackErr;
	char pLastRecStr[BUF_SIZE];
};
```

6.1.1 Constructores de la clase UCom

Los constructores son funciones miembro especiales que sirven para inicializar un objeto de una determinada clase al mismo tiempo que se declara. Los constructores tienen el mismo nombre que la clase a la que pertenecen, no tienen tipo de retorno, y no pueden ser heredados. Por último, deben ser públicos, ya que siempre se usan desde el exterior de la clase, ni pueden ser protegidos, ya que no pueden ser heredados.

```
UCom::UCom()
{
	hCommDev=INVALID_HANDLE_VALUE;
	bDebugMode=false;
	bCheckCrcMode = false;
	bCheckLineMode = false;
	bCheckReadAll = false;
	uLastNackErr=0;
	strncpy(pLastRecStr,"",sizeof(pLastRecStr));

	pUCom=this;
}


////////////////////////////////////////////////////////////////////////////////
// Función: UCom::~UCom
// Descripción: Destructor de clase
////////////////////////////////////////////////////////////////////////////////
UCom::~UCom()
{
}
```

Se destacan a continuación las funciones más importantes implementadas dentro de la clase. Se incluyen de esta forma funciones para la conexión, envío y recepción, etc.

Tabla 10: Principales funciones de la clase UCom

Función	Descripción
UCom::Connection	Asignación del puerto serie (miembro de Ucom) al habdle global
UCom::Write	Envía una buffer de datos por el puerto serie
UCom::Read	Lee del puerto serie y almacena los datos en un buffer. Si no hay datos, retorna.
UCom::SendString	Envía por puerto serie una cadena hasta el terminador
UCom::ReceiveString	Espera la recepción de una cadena hasta el terminador
UCom::CheckForAck	Espera la recepción de un asentimiento mediante ReceiveString.
UCom::PrintNackErr	Devuelve una cadena de error para interpretar un asentimiento negativo

```
////////////////////////////////////////////////////////////////////////////
// Función: UCom::Connection
// Descripción: Asignación del puerto serie (miembro de Ucom) al habdle global
////////////////////////////////////////////////////////////////////////////
BOOL UCom::Connection()
{
	hCommDev=puertoserie; //Copia la variable global en local
	return true;
}



////////////////////////////////////////////////////////////////////////////
// Función: UCom::Write
// Descripción: Envía una buffer de datos por el puerto serie.
// Recibe: puntero al inicio del buffer, longitud
// Devuelve: Resultado de la operación, true o false
////////////////////////////////////////////////////////////////////////////
BOOL UCom::Write(char *lpBuffer,int len)
{
int NumOfBytesToWrite;
	if(!len)
		NumOfBytesToWrite=strlen(lpBuffer);
		else
```

```
		NumOfBytesToWrite=len;

		DWORD NumberOfBytesWritten;
		BOOL bOk=WriteFile
		(
			hCommDev,		// handle to file to write to
			lpBuffer,		// pointer to data to write to file
			NumOfBytesToWrite,//strlen(lpBuffer),	// bytes to write
			&NumberOfBytesWritten,	// pointer to number of bytes written
			NULL	// pointer to structure for overlapped I/O
		);
		if(!bOk)
		{
			ErrorMessage();
			return false;
		}
	return true;
}


////////////////////////////////////////////////////////////////////////////////
// Función: UCom::Read
// Descripción: Lee del puerto serie y almacena los datos en un buffer. Si no hay datos, retorna.
// Recibe: puntero al inicio del buffer, longitud
// Devuelve: Resultado de la operación, true o false
////////////////////////////////////////////////////////////////////////////////
DWORD UCom::Read(char *lpBuffer,DWORD nNumberOfBytesToRead)
{
	lpBuffer[0]=0;
	DWORD lNumberOfBytesRead =0;

	if(hCommDev!=INVALID_HANDLE_VALUE)
	{

		BOOL bOk=ReadFile
		(
			hCommDev,		// handle of file to read
			lpBuffer,		// pointer to buffer that receives data
			nNumberOfBytesToRead,	// number of bytes to read
			&lNumberOfBytesRead, // pointer to number of bytes read
			NULL	// pointer to structure for data
		);
		if(!bOk)
		{
			ErrorMessage();
```

```
                        return 0;
                }
        }
        if(lNumberOfBytesRead<0)
        {
                lpBuffer[0]=0;
                ErrorMessage();
                return 0;
        }
        lpBuffer[lNumberOfBytesRead]=0;

        return lNumberOfBytesRead;
}



////////////////////////////////////////////////////////////////////////////////
// Función: UCom::SendString
// Descripción: Envía por puerto serie una cadena hasta el terminador
////////////////////////////////////////////////////////////////////////////////
BOOL UCom::SendString(char *lpBuffer)
{
        char Buf1[BUF_SIZE];
        char Buf2[BUF_SIZE];
        Buf1[0]=0;
        Buf2[0]=0;

        if(strlen(lpBuffer)>BUF_SIZE_1)
        {
                LogBox("Send Buffer Overflow");
                return false;
        }

        strcpy(Buf1,lpBuffer);
        if(!bCheckCrcMode)   sprintf(Buf2,"\n");
        else                            sprintf(Buf2,"*%02X\n",GetStrCRC8(Buf1));

        if(strlen(Buf1)+strlen(Buf2)>BUF_SIZE_1)
        {
                LogBox("Send   Buffer   Overflow",   "UCom   Error",MB_TASKMODAL   |
MB_OK | MB_ICONERROR );
                return false;
        }

        strcat(Buf1,Buf2);
        return Write(Buf1);
}
```

```
////////////////////////////////////////////////////////////////////////////////
// Función: UCom::ReceiveString
// Descripción: Espera la recepción de una cadena hasta el terminador
// Recibe: Buffer de entrada, numero de bytes por leer y booleano (1=comprobar Ack, 0=No comprobar)
// Devuelve: Resultado de la comprobación del asentimiento
////////////////////////////////////////////////////////////////////////////////
BOOL UCom::ReceiveString(char *lpBuffer, DWORD nNumberOfBytesToRead, BOOL bCheckForAck)
{
DWORD lDataReady=0;
int Endmark=0;
int ixx=0;
long itime=0;


	uLastNackErr=0; //Reset uLastNackErr

				Read(lpBuffer,nNumberOfBytesToRead);
				return true;
	strncpy(pLastRecStr,lpBuffer,sizeof(pLastRecStr)-1);

	if(*lpBuffer==CH_NACK)
	{
		if(sscanf(lpBuffer+1,"%X",&uLastNackErr)!=1) //Set uLastNackErr
		{
			if(MessageBox(NULL,"NACK without error code is not found", "UCom NACK",MB_TASKMODAL | MB_OKCANCEL | MB_ICONWARNING )==IDCANCEL)
			return false;
		}
	}
	else if(bCheckForAck && *lpBuffer!=CH_ACK)
	{
		if(MessageBox(NULL,lpBuffer, "UCom Missing ACK",MB_TASKMODAL | MB_OKCANCEL | MB_ICONWARNING )==IDCANCEL)
			return false;
	}

	if(bCheckCrcMode)
	{
		char *pCRC=lpBuffer;
		while(*pCRC!='*')
		{
			if(!(*pCRC))
```

```
                        {
                                if(MessageBox(NULL,"CRC is not found", "UCom CRC",MB_TASKMODAL | MB_OKCANCEL | MB_ICONWARNING )==IDCANCEL)
                                return false;
                        }
                        pCRC++;
                }
                *pCRC=0;//Cud the string
                pCRC++;//Point the pCRC to CRC suffiks

                WORD wCRC1,wCRC2;
                wCRC1=GetStrCRC8(lpBuffer);
                if(sscanf(pCRC,"%2X",&wCRC2)!=1 || wCRC1!=wCRC2)
                {
                        if(MessageBox(NULL,"CRC Error", "UCom CRC",MB_TASKMODAL | MB_OKCANCEL | MB_ICONWARNING )==IDCANCEL)
                        return false;
                }
                strcat(lpBuffer,"\r\n");
        }

        return true;
}



////////////////////////////////////////////////////////////////////////////////
// Función: UCom::CheckForAck
// Descripción: Espera la recepción de un asentimiento mediante ReceiveString.
////////////////////////////////////////////////////////////////////////////////
BOOL UCom::CheckForAck()
{
        char lpRspBuf[BUF_SIZE];
        return ReceiveString(lpRspBuf,BUF_SIZE_1,true);
}



////////////////////////////////////////////////////////////////////////////////
// Función: UCom::PrintNackErr
// Descripción: Devuelve una cadena de error para interpretar un asentimiento negativo
////////////////////////////////////////////////////////////////////////////////
char *UCom::PrintNackErr(UINT uErr)
{
        switch(uErr)
        {
```

```
                case UNDEF_0_ERR:   return "UNDEF_0_ERR";
                case UNDEF_1_ERR:   return "UNDEF_1_ERR";
                case ILL_CHR_ERR:     return "ILL_CHR_ERR";
                case BUF_OVF_ERR:   return "BUF_OVF_ERR";
                case CTL_CHR_ERR:    return "CTL_CHR_ERR";
                case DOM_ERR:                 return "DOM_ERR";
                case WRP_ERR:                 return "WRP_ERR";
                case NOT_REM_ERR: return "NOT_REM_ERR";
                case NOT_SBY_ERR:   return "NOT_SBY_ERR";
                case OUT_BND_ERR: return "OUT_BND_ERR";
                case CMD_CMB_ERR:return "CMD_CMB_ERR";
                case CRC_ERR:                 return "CRC_ERR";
                case TIMING_ERR:      return "TIMING_ERR";
                case NOT_IN_AOFF:   return "NOT_IN_AOFF";
                case BUF_LEN_ERR:   return "BUF_LEN_ERR";
                case CFG_ACHAN_ERR:         return "CFG_ACHAN_ERR";
                default:                        return "Unknown Error";
        }
}
```

6.2 Implementación del protocolo Buit-In-Test

Para el manejo del protocolo BIT, de Orbit, se ha programado la clase BitMonitor1, en la que se han implementado los métodos para la comunicación con las entidades BIT de los DSA.

Entre las funcionalidades más destacadas se encuentran métodos para el envío y recepción de SDO, evaluación de mensajes o actualización de configuraciones.

```
class BitMonitor1
{
// Construction
public:
        float time;
        unsigned int Rate;
        void RdSdo(unsigned int WrIndex, char WrSub, char DataType );
        char Fname [15];
        void EvalMessage(char BuffOut[]);
        void WrSdo(unsigned int WrIndex,char WrSub,char DataType,char *str);
        void UpdateSettings();
        void fin();
        BOOL inicio() ;
        void OnClrBit();
        void OnKillfocusUpdRate();
        BOOL Create(int Id);
```

```
// Implementation
protected:
        unsigned long T2;
        unsigned long T1;
        int IndexSel[4];
        int Ilog;
        int Log2File;
        FILE * flog;
//Wnd* m_pParent;
        int m_nID;
   int Id;
};
```

6.2.1 Constructores de la clase BitMonitor1

Se listan a continuación los constructores más destacados de la clase.

Tabla 11: Funciones principales de la clase BitMonitor1

Funciones	Descripción
BitMonitor1::Create	Inicialización de la clase Bitmonitor
BitMonitor1::fin	Paso previo a la salida del modo BIT. Envía SDO para cancelar el envío periódico del BIT
BitMonitor1::inicio	Paso previo a la entrada del modo BIT. Envía SDO para iniciar el envío periódico del BIT
BitMonitor1::WrSdo	Envío de tramas SDO genéricas para escribir palabras del Object Dictionary
BitMonitor1::RdSdo	Envío de SDO ara solicitar el valor de un objeto del Object Dictionary que será devuelto por vía SDO

```
////////////////////////////////////////////////////////////////////////////
// Función: BitMonitor1::Create
// Descripción: Inicialización de la clase Bitmonitor
// Argumentos: Identificación de eje (1=acimut 0=elevación)
// Devuelve: true
////////////////////////////////////////////////////////////////////////////
BOOL BitMonitor1::Create(int WinId)
{
        Id=WinId;
        if(Id==AZ_AX)
                strcpy(Fname,"AzBit.cfg");
```

```
                else
                strcpy(Fname,"ElBit.cfg");
                Log2File=0;
                flog=NULL;
        return true;

}



////////////////////////////////////////////////////////////////////////////////
// Función: BitMonitor1::fin
// Descripción: Paso previo a la salida del modo BIT. Envía SDO para cancelar el envío
// periódico del BIT
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void BitMonitor1::fin()
{
char str[50];
unsigned int WrIndexBit;
char WrSubBit,DataType;

    if(Connected2B)
                WrIndexBit=PDO_MAP_INDEX+4;
                else
                WrIndexBit=PDO_MAP_INDEX;
                WrSubBit=0x00;
                        WrIndex=WrIndexBit;
                        WrSub=WrSubBit;
                DataType=UINT8;
                sprintf(str,"0");
                WrSdo(WrIndex,WrSub,DataType,str);
}



////////////////////////////////////////////////////////////////////////////////
// Función: BitMonitor1::inicio
// Descripción: Paso previo a la entrada del modo BIT. Envía SDO para iniciar el envío
// periódico del BIT
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
BOOL BitMonitor1::inicio()
{
char resp[50];
```

```
FILE *fp;
int FldIx1,FldIx2,FldIx3,FldIx4,FldIx5;
        if(Id==AZ_AX)
        sprintf(resp, "Azimuth BitMonitor  " );
        else
        sprintf(resp, "Elevation BitMonitor  " );
        LogBox(resp);
        FldIx1=FldIx2=FldIx3=FldIx4=FldIx5=0;
        fp=fopen(Fname,"r");
        if(fp!=NULL)
        {
                fscanf(fp,"%d %d %d %d %d",&FldIx1,&FldIx2,&FldIx3,&FldIx4,&FldIx5);
                fclose(fp);
                fp=NULL;
        }
        IndexSel[0]=FldIx1;
        IndexSel[1]=FldIx2;
        IndexSel[2]=FldIx3;
        IndexSel[3]=FldIx4;
        time=0;
        T1=T2=0;
        flog=NULL;
        UpdateSettings();
        return TRUE;
}



////////////////////////////////////////////////////////////////////////////////
// Función: BitMonitor1::WrSdo
// Descripción: Envío de tramas SDO genéricas para escribir palabras del Object Dictionary
// Argumentos: Indice, subíndice, tipo y valor (en forma de cadena de caracteres) del dato a escribir
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void BitMonitor1::WrSdo(unsigned int WrIndex, char WrSub, char DataType, char *str)
{

//CString str;
char BuffOut[22];
int len;
//float SlewCom;
unsigned int CobId,n;
unsigned long udata32;
short sdata8;
long sdata32;
```

```
unsigned short udata8;
unsigned int udata16;
int sdata16;
float rl32;
FlotArr Flot2Bytes;


                BuffOut[6]=0;
                BuffOut[7]=0;
                BuffOut[8]=0;
                BuffOut[9]=0;
                switch(DataType)
                {
                        case UINT32:
                                n=4;
                                sscanf(str,"%lu",&udata32);
                                BuffOut[9]=(char)((udata32&0xff000000)>>24);
                                BuffOut[8]=(char)((udata32&0x00ff0000)>>16);
                                BuffOut[7]=(char)((udata32&0x0000ff00)>>8);
                                BuffOut[6]=(char)((udata32&0x000000ff));
                                break;
                        case UINT16:
                                n=2;
                                sscanf(str,"%u",&udata16);
                                BuffOut[7]=(char)((udata16&0x0000ff00)>>8);
                                BuffOut[6]=(char)((udata16&0x000000ff));
                                break;
                        case UINT8:
                                n=1;
                                sscanf(str,"%u",&udata8);
                                BuffOut[6]=(char)((udata8&0x000000ff));
                                break;
                        case INT32:
                                n=4;
                                sscanf(str,"%ld",&sdata32);
                                BuffOut[9]=(char)((sdata32&0xff000000)>>24);
                                BuffOut[8]=(char)((sdata32&0x00ff0000)>>16);
                                BuffOut[7]=(char)((sdata32&0x0000ff00)>>8);
                                BuffOut[6]=(char)((sdata32&0x000000ff));
                                break;
                        case INT16:
                                n=2;
                                sscanf(str,"%hd",&sdata16);
                                BuffOut[7]=(char)((sdata16&0x0000ff00)>>8);
                                BuffOut[6]=(char)((sdata16&0x000000ff));
                                break;
                        case INT8:
```

```
                        n=1;
                        sscanf(str,"%hd",&sdata8);
                        BuffOut[6]=(char)((sdata8&0x000000ff));
                        break;
                case RL32:
                        n=4;
                                sscanf(str,"%f",&rl32);
                                Flot2Bytes.fnr=rl32;
                                BuffOut[6]=Flot2Bytes.binnr[0];
                                BuffOut[7]=Flot2Bytes.binnr[1];
                                BuffOut[8]=Flot2Bytes.binnr[2];
                                BuffOut[9]=Flot2Bytes.binnr[3];
                        break;
                }

                if(Id==AZ_AX)
        CobId=SDO_RX|AZ_COB_ID;
                else
                CobId=SDO_RX|EL_COB_ID;
                CobId|=0x0008; // 8 bytes data len
                BuffOut[0]=CobId>>8;
                BuffOut[1]=CobId&0x00ff;
                BuffOut[2]=0x23; // bits 7-5 csc=1,e=1,s=1;
                BuffOut[2]|=(4-n)<<2;// insert n
                BuffOut[3]=(char)((WrIndex&0x000000ff));
                BuffOut[4]=((WrIndex&0x0000ff00)>>8);
                BuffOut[5]=(char)WrSub;


                len=10;
                SendMsg( BuffOut ,  len); //fgs

}



////////////////////////////////////////////////////////////////////////////////
// Función: BitMonitor1::RdSdo
// Descripción: Envío de SDO ara solicitar el valor de un objeto del Object Dictionary
// que será devuelto por vía SDO
// Argumentos: Indice, subíndice y tipo de dato a leer
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void BitMonitor1::RdSdo(unsigned int RdIndex, char RdSub, char DataType)
{

char BuffOut[22];
```

```
int len;
unsigned int CobId ;

                if(Id==AZ_AX)
        CobId=SDO_RX|AZ_COB_ID;
                else
                CobId=SDO_RX|EL_COB_ID;
                CobId|=0x0008;
                BuffOut[0]=CobId>>8;
                BuffOut[1]=CobId&0x00ff;
                BuffOut[2]=0x40;
                BuffOut[3]=(char)((RdIndex&0x000000ff));
                BuffOut[4]=((RdIndex&0x0000ff00)>>8);
                BuffOut[5]=(char)RdSub;
                BuffOut[6]=0;
                BuffOut[7]=0;
                BuffOut[8]=0;
                BuffOut[9]=0;
                len=10;
                SendMsg( BuffOut ,  len);

}
```

6.3 Puntos de entrada de la DLL

En este apartado ya se ha comentado la manera en que es llamada la librería para poder acceder a sus funcionalidades. Encontramos de esta forma cuatro funciones para el manejo de los flujos de información de entrada y salida a otras aplicaciones.

La primera de ellas es la función DLL, que será la que se invoque en el modo de funcionamiento normal. Dicha función mapea los comandos del usuario y llama a las funciones que implementen los métodos deseados.

```
////////////////////////////////////////////////////////////////////////////////
// Función: DLL
// Descripción: Punto de entrada principal a la DLL: Mapea los comandos del usuario
// Consultar el manual de usuario
// Argumentos: Punteros a estructras, de tipos variables dependiendo del comande que se trate
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
int __stdcall DLL(void * ig1, void * ig2, void * ig3, long Modo_Op, long Comando)
{
```

```
switch(Modo_Op){
case MODO_PDO:
        VControl(ig1,ig2,ig3,Comando);
        break;

case MODO_BIT:
        VBit(ig1,Comando);
        break;

case MODO_SDO:
        return (VSDO(ig1,Comando));
        break;

case MODO_SEARCH_CFG:

        IntGrafSrch=(graficosSRCH * )ig1;


        if(Comando==EVENTO_SEARCH_SETUP_LEER_FICHERO)
        {       // Leemos stup.prm y actualizamos la estructura global y la estructura de busqueda de VB

                FILE *fp;
                fp=fopen("SetUp.prm","r");
                if(fp!=NULL)
                {       fscanf(fp,"%f %f %f %f %f %f %f %f %d %d\n",&Srch.m_az_cen,&Srch.m_az_sec,&Srch.m_az_stp,&Srch.m_az_vel,&Srch.m_el_cen,&Srch.m_el_sec,&Srch.m_el_stp,&Srch.m_el_vel,&Srch.Type,&Srch.UpdVia);
                        fclose(fp);
                        *IntGrafSrch=Srch;
                }
                else    LogBox("No se ha podido abrir el fichero setup.prm");
        }
        else{ /* hay que actualizar la estructura seacrh con los datos pasados como parametro */
                Srch=*IntGrafSrch;
        FILE *fp;
        fp=fopen("SetUp.prm","w");
        if(fp==NULL){
                LogBox("No se ha podido abrir el fichero setup.prm");
                return 0;
        }
        fprintf(fp,"%7.3f %7.3f %7.3f %7.3f %7.3f %7.3f %7.3f %7.3f %d %d\n",Srch.m_az_cen,Srch.m_az_sec,Srch.m_az_stp,Srch.m_az_vel,Srch.m_el_cen,Srch.m_el_sec,Srch.m_el_stp,Srch.m_el_vel,Srch.Type,Srch.UpdVia);
        fclose(fp);
```

```
        // Comunicar a la DSA la nueva configuración
                OnSearch();
                };
                break;
        }

return 0;
}
```

Las dos siguientes funciones son las de inicio y fin de la aplicación. Abren o cierran el puerto usado, según sea el caso y finDLL borra las clases de BitMonitor para acimut y elevación.

```
////////////////////////////////////////////////////////////////////////////////
// Función: finDLL
// Descripción: Punto de entrada a la DLL: Paso previo a la Finalización de la aplicación
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////

int  __stdcall finDLL(void)
{
        AzBitMonitor->fin();
        ElBitMonitor->fin();
        delete ElBitMonitor;
        delete AzBitMonitor;
        finlog();
        if (puertoserie!=NULL) CloseHandle(puertoserie);
        return 1;
}
```

```
////////////////////////////////////////////////////////////////////////////////
// Función: inicioDLL
// Descripción: Punto de entrada a la DLL: Inicialización con un handle de puerto ya abierto
// Argumentos: entero largo HANDLE del puerto serie abierto
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
int __stdcall inicioDLL(long puerto)
{
        inicio(puerto);
        return 1;
}
```

Finalmente, se encuentra la función para iniciar la DLL si el puerto aún no ha sido abierto.

```
////////////////////////////////////////////////////////////////////////////
// Función: inicioDLLport
// Descripción: Punto de entrada a la DLL: Inicialización sin puerto abierto.
// Crea el handle y llama a la función de inicio con puerto ya creado
// Argumentos: estructura con los datos del puerto que se desea abrir
////////////////////////////////////////////////////////////////////////////
int __stdcall inicioDLLport(void * param)
{
	TipoPuertoConf * PuertoConf;
	HANDLE puertoserielocal=INVALID_HANDLE_VALUE;
	BOOL bOk;
	char cadenapuerto[10];


// Apertura puerto con la configuración deseada
	PuertoConf=(TipoPuertoConf *)param;
	sprintf(cadenapuerto,"COM%d",PuertoConf->numeroCOM);
	puertoserielocal = CreateFile(cadenapuerto,
				GENERIC_READ | GENERIC_WRITE,
				NULL,
				NULL,
				OPEN_EXISTING,
				FILE_ATTRIBUTE_NORMAL,
				NULL);

		if (puertoserielocal==INVALID_HANDLE_VALUE) return 0;
		bOk = GetCommState(puertoserielocal, &MyDCB);
		if (!bOk) return 0;

//Reconfigurar COM según las propiedades de la estructura DCB modificada.
		MyDCB.BaudRate = PuertoConf->baudios;
		MyDCB.ByteSize = PuertoConf->bitsdatos;
		MyDCB.Parity = PuertoConf->paridad;
		MyDCB.StopBits = PuertoConf->bitsparada;

		bOk = SetCommState(puertoserielocal, &MyDCB);
		if (!bOk) return 0;

//Recuperar la configuración actual de tiempo de espera.
		bOk = GetCommTimeouts(puertoserielocal, &CT);

		CT.ReadIntervalTimeout=MAXDWORD;
		CT.ReadTotalTimeoutMultiplier=0;
```

```
		CT.ReadTotalTimeoutConstant=0;
		CT.WriteTotalTimeoutMultiplier=0;
		CT.WriteTotalTimeoutConstant=0;

		bOk=SetCommTimeouts
		(
			puertoserielocal,		// handle to comm device
			&CT  // pointer to comm time-out structure
		);
		if (!bOk) return 0;

		inicio((long) puertoserielocal);

		return 1;
}
```

6.4 Funciones para el manejo del protocolo OCP

La implementación del protocolo OCP, explicado en el apartado anterior, requiere un complejo tratamiento. Las funciones más importantes a la hora de realizar dicha tarea se listan a continuación.

Tabla 12: Funciones del protocolo OCP

Función	Descripción
SendPDO	Envío inmediato de PDO para comandos de cambio de modo y referencias de posición y velocidad.
SendRSDO	Envío de SDO para lectura de palabras del object dictionary. Modifica la estructura InterfazGraficaSDO
SendWSDO	Envío de SDO para escritura de palabras del object dictionary. Emplea la estructura InterfazGraficaSDO
WrSdo	Envío de SDOs para escribir siobre una palabra del object dictionary

```
////////////////////////////////////////////////////////////////////////////////
// Función: SendPDO
```

```
// Descripción: envío inmediato de PDO para comandos de cambio de modo y referencias de posición y velocidad.
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  SendPDO()
{
 char BuffOut[22];
int len;

unsigned int iPoint,iSrch;
unsigned int CobId;
int iVel;
 FlotArr Flot2Bytes;

        if(!TrnsPDO) return;
        if (TrnsPDO&AZ_AX)
        {
                //Send Search Via PDO
                if(Srch.UpdVia==VIA_PDO)
                { //Send Pdo 2
                        iSrch=(unsigned int)( Srch.m_az_sec*(float)DEG2FIX);
                CobId=PDO2_RX|AZ_COB_ID;//AZ_COB_ID;
                        CobId|=0x0008; // 8 bytes data len
                        BuffOut[0]=CobId>>8;
                        BuffOut[1]=CobId&0x00ff;
                        BuffOut[3]=(iSrch&0xff00)>>8;
                        BuffOut[2]=iSrch&0x00ff;
                        iSrch=(unsigned int)( Srch.m_az_stp*(float)DEG2FIX);
                        BuffOut[5]=(iSrch&0xff00)>>8;
                        BuffOut[4]=iSrch&0x00ff;
                        Flot2Bytes.fnr=Srch.m_az_vel;
                        BuffOut[6]=Flot2Bytes.binnr[0];
                        BuffOut[7]=Flot2Bytes.binnr[1];
                        BuffOut[8]=Flot2Bytes.binnr[2];
                        BuffOut[9]=Flot2Bytes.binnr[3];
                        len=10;
                        len=Ocp.CompMsg(0,BuffOut,len);
        theUCom.Write(BuffOut, len);
                }
                iPoint=(unsigned int)( PosCom*(float)DEG2FIX);
                iVel=(int) (SlewCom*(float)VEL_DEG2FIX);
        CobId=PDO1_RX|AZ_COB_ID;//AZ_COB_ID;
                if(Srch.UpdVia==VIA_PDO)
                CobId|=0x0008; // 8 bytes data len
                else
                CobId|=0x0005; // 5 bytes data len
```

```
                BuffOut[0]=CobId>>8;
                BuffOut[1]=CobId&0x00ff;
                BuffOut[2]=(char)Mode;
                BuffOut[4]=(iPoint&0xff00)>>8;
                BuffOut[3]=iPoint&0x00ff;
                BuffOut[6]=(iVel&0xff00)>>8;
                BuffOut[5]=iVel&0x00ff;
                len=7;
                if(Srch.UpdVia==VIA_PDO)
                {
                        iSrch=(unsigned int)( Srch.m_az_cen*(float)DEG2FIX);
                        BuffOut[8]=(iSrch&0xff00)>>8;
                        BuffOut[7]=iSrch&0x00ff;
                        BuffOut[9]=Srch.Type;
                        len=10;
                }
                len=Ocp.CompMsg(0,BuffOut,len);
                theUCom.Write(BuffOut, len);
        }
        if (TrnsPDO&EL_AX)
        {
                if(Srch.UpdVia==VIA_PDO)
                { //Send Pdo 2
                        iSrch=(unsigned int)( Srch.m_el_sec*(float)DEG2FIX);
                CobId=PDO2_RX|EL_COB_ID;//AZ_COB_ID;
                        CobId|=0x0008; // 8 bytes data len
                        BuffOut[0]=CobId>>8;
                        BuffOut[1]=CobId&0x00ff;
                        BuffOut[3]=(iSrch&0xff00)>>8;
                        BuffOut[2]=iSrch&0x00ff;
                        iSrch=(unsigned int)( Srch.m_el_stp*(float)DEG2FIX);
                        BuffOut[5]=(iSrch&0xff00)>>8;
                        BuffOut[4]=iSrch&0x00ff;
                        Flot2Bytes.fnr=Srch.m_el_vel;
                        BuffOut[6]=Flot2Bytes.binnr[0];
                        BuffOut[7]=Flot2Bytes.binnr[1];
                        BuffOut[8]=Flot2Bytes.binnr[2];
                        BuffOut[9]=Flot2Bytes.binnr[3];
                        len=10;
                        len=Ocp.CompMsg(0,BuffOut,len);
                        theUCom.Write(BuffOut, len);
                }
//**********************
                iPoint=(unsigned int)( PosCom2*(float)DEG2FIX);
                iVel=(int) (SlewCom2*(float)VEL_DEG2FIX);
        CobId=PDO1_RX|EL_COB_ID;
                if(Srch.UpdVia==VIA_PDO)
```

```
                CobId|=0x0008; // 8 bytes data len
                else
                CobId|=0x0005; // 5 bytes data len
                BuffOut[0]=CobId>>8;
                BuffOut[1]=CobId&0x00ff;
                BuffOut[2]=(char)Mode2;
                BuffOut[4]=(iPoint&0xff00)>>8;
                BuffOut[3]=iPoint&0x00ff;
                BuffOut[6]=(iVel&0xff00)>>8;
                BuffOut[5]=iVel&0x00ff;
                len=7;
                if(Srch.UpdVia==VIA_PDO)
                {
                        iSrch=(unsigned int)( Srch.m_el_cen*(float)DEG2FIX);
                        BuffOut[8]=(iSrch&0xff00)>>8;
                        BuffOut[7]=iSrch&0x00ff;
                        BuffOut[9]=Srch.Type;
                        len=10;
                }
                len=Ocp.CompMsg(0,BuffOut,len);
                theUCom.Write(BuffOut, len);
        }
                TrnsPDO=0;

}



////////////////////////////////////////////////////////////////////////////////
// Función: SendRSDO
// Descripción: envío de SDO para lectura de palabras del object dictionary. Modifica la
estructura InterfazGraficaSDO
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  SendRSDO()
{
char BuffOut[22];
int len;
//float SlewCom;
unsigned int CobId;

        if(!TrnsRSDO) return;
        TrnsRSDO=0;

        CobId=SDO_RX|NodeID;
                CobId|=0x0008; // 8 bytes data len
```

```
                BuffOut[0]=CobId>>8;
                BuffOut[1]=CobId&0x00ff;
                BuffOut[2]=0x40;
                BuffOut[3]=(char)((RdIndex&0x000000ff));
                BuffOut[4]=((RdIndex&0x0000ff00)>>8);
                BuffOut[5]=(char)RdSub;
                BuffOut[6]=0;
                BuffOut[7]=0;
                BuffOut[8]=0;
                BuffOut[9]=0;
                len=10;
                len=Ocp.CompMsg(0,BuffOut,len);
                theUCom.Write(BuffOut, len);

}


////////////////////////////////////////////////////////////////////////////////
// Función: SendWSDO
// Descripción: envío de SDO para escritura de palabras del object dictionary. Emplea
la estructura InterfazGraficaSDO
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  SendWSDO()
{
char str[100];
//char resp[100];

char BuffOut[22];
int len;
//float SlewCom;
unsigned int CobId,n;
unsigned long udata32;
short sdata8;
long sdata32;
unsigned short udata8;
unsigned int udata16;
int sdata16;

short DataFrmt;

        if(!TrnsWSDO) return;
        if (1)
        DataFrmt=HEX;
        else
```

```
        DataFrmt=DECIMAL;
//********
        if(SetUpInPrgrs)
        {
                strcpy(str,&value[FileIndex][0]);
                DataFrmt=DECIMAL;
        }
        else
        TrnsWSDO=0;

        CobId=SDO_RX|NodeID;//AZ_COB_ID;
                CobId|=0x0008; // 8 bytes data len
                BuffOut[0]=CobId>>8;
                BuffOut[1]=CobId&0x00ff;
                BuffOut[2]=0x23; // bits 7-5 csc=1,e=1,s=1;
                ASSERT(InterfazGraficaSDO!=NULL);
                n=GetDataType();
                BuffOut[2]|=(4-n)<<2;// insert n
                BuffOut[3]=(char)((WrIndex&0x000000ff));
                BuffOut[4]=((WrIndex&0x0000ff00)>>8);
                BuffOut[5]=(char)WrSub;
                BuffOut[6]=0;
                BuffOut[7]=0;
                BuffOut[8]=0;
                BuffOut[9]=0;
                switch(DataType)
                {
                        case UINT32:
                                udata32=(__int32) InterfazGraficaSDO->ValorEscribir;
                                BuffOut[9]=(char)((udata32&0xff000000)>>24);
                                BuffOut[8]=(char)((udata32&0x00ff0000)>>16);
                                BuffOut[7]=(char)((udata32&0x0000ff00)>>8);
                                BuffOut[6]=(char)((udata32&0x000000ff));
                                break;
                        case UINT16:
                                udata16=(__int16) InterfazGraficaSDO->ValorEscribir;
                                BuffOut[7]=(char)((udata16&0x0000ff00)>>8);
                                BuffOut[6]=(char)((udata16&0x000000ff));
                                break;
                        case UINT8:
                                BuffOut[6]=(char)((udata8&0x000000ff));
                                break;
                        case INT32:
                                sdata32=(__int32) InterfazGraficaSDO->ValorEscribir;
                                BuffOut[9]=(char)((sdata32&0xff000000)>>24);
                                BuffOut[8]=(char)((sdata32&0x00ff0000)>>16);
                                BuffOut[7]=(char)((sdata32&0x0000ff00)>>8);
```

```
                    BuffOut[6]=(char)((sdata32&0x000000ff));
                    break;
                case INT16:
                    sdata16=(__int16) InterfazGraficaSDO->ValorEscribir;
                    BuffOut[7]=(char)((sdata16&0x0000ff00)>>8);
                    BuffOut[6]=(char)((sdata16&0x000000ff));
                    break;
                case INT8:
                    sdata8=(__int8) InterfazGraficaSDO->ValorEscribir;
                    BuffOut[6]=(char)((sdata8&0x000000ff));
                    break;
                case RL32:
                    udata32=(__int32) InterfazGraficaSDO->ValorEscribir;
                    BuffOut[9]=(char)((udata32&0xff000000)>>24);
                    BuffOut[8]=(char)((udata32&0x00ff0000)>>16);
                    BuffOut[7]=(char)((udata32&0x0000ff00)>>8);
                    BuffOut[6]=(char)((udata32&0x000000ff));

            }

            len=10;
            len=Ocp.CompMsg(0,BuffOut,len);
            theUCom.Write(BuffOut, len);

}



////////////////////////////////////////////////////////////////////////////////
// Función: WrSdo
// Descripción: Envío de SDOs para escribir siobre una palabra del object dictionary
// Argumentos: los de la SDO (índice, subíndice, tipo de dato y contenido)
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  WrSdo(unsigned int WrIndex, char WrSub, char DataType, char *str,int Id)
{

//CString str;
char BuffOut[22];
int len;
unsigned int CobId,n;
unsigned long udata32;
short sdata8;
long sdata32;
unsigned short udata8;
unsigned int udata16;
int sdata16;
```

```
float rl32;
FlotArr Flot2Bytes;


        BuffOut[6]=0;
        BuffOut[7]=0;
        BuffOut[8]=0;
        BuffOut[9]=0;
        switch(DataType)
        {
                case UINT32:
                        n=4;
                        sscanf(str,"%lu",&udata32);
                        BuffOut[9]=(char)((udata32&0xff000000)>>24);
                        BuffOut[8]=(char)((udata32&0x00ff0000)>>16);
                        BuffOut[7]=(char)((udata32&0x0000ff00)>>8);
                        BuffOut[6]=(char)((udata32&0x000000ff));
                        break;
                case UINT16:
                        n=2;
                        sscanf(str,"%u",&udata16);
                        BuffOut[7]=(char)((udata16&0x0000ff00)>>8);
                        BuffOut[6]=(char)((udata16&0x000000ff));
                        break;
                case UINT8:
                        n=1;
                        sscanf(str,"%u",&udata8);
                        BuffOut[6]=(char)((udata8&0x000000ff));
                        break;
                case INT32:
                        n=4;
                        sscanf(str,"%ld",&sdata32);
                        BuffOut[9]=(char)((sdata32&0xff000000)>>24);
                        BuffOut[8]=(char)((sdata32&0x00ff0000)>>16);
                        BuffOut[7]=(char)((sdata32&0x0000ff00)>>8);
                        BuffOut[6]=(char)((sdata32&0x000000ff));
                        break;
                case INT16:
                        n=2;
                        sscanf(str,"%hd",&sdata16);
                        BuffOut[7]=(char)((sdata16&0x0000ff00)>>8);
                        BuffOut[6]=(char)((sdata16&0x000000ff));
                        break;
                case INT8:
                        n=1;
                        sscanf(str,"%hd",&sdata8);
                        BuffOut[6]=(char)((sdata8&0x000000ff));
```

```
                    break;
                case RL32:
                    n=4;
                        sscanf(str,"%f",&rl32);
                        Flot2Bytes.fnr=rl32;
                        BuffOut[6]=Flot2Bytes.binnr[0];
                        BuffOut[7]=Flot2Bytes.binnr[1];
                        BuffOut[8]=Flot2Bytes.binnr[2];
                        BuffOut[9]=Flot2Bytes.binnr[3];
                    break;
            }

            if(Id==AZ_AX)
        CobId=SDO_RX|AZ_COB_ID;
            else
            CobId=SDO_RX|EL_COB_ID;
            CobId|=0x0008; // 8 bytes data len
            BuffOut[0]=CobId>>8;
            BuffOut[1]=CobId&0x00ff;
            BuffOut[2]=0x23; // bits 7-5 csc=1,e=1,s=1;
            BuffOut[2]|=(4-n)<<2;// insert n
            BuffOut[3]=(char)((WrIndex&0x000000ff));
            BuffOut[4]=((WrIndex&0x0000ff00)>>8);
            BuffOut[5]=(char)WrSub;
            len=10;
            SendMsg( BuffOut ,  len);

}
```

6.5 Funciones para el manejo de los modos de funcionamiento

Finalmente, se presentan las funciones que implementan los distintos modos de funcionamiento de los DSA. Como ya se ha comentado en el presente documento, existen diferentes modos de funcionamiento según las necesidades del usuario. Se pueden realizar actuaciones en posición o velocidad, paradas y estados de stand by.

A continuación se listan las funciones más importantes en este entorno:

Tabla 13: Funciones principales para el manejo de los modos

Funciones	Descripción
OnSlew	Cambio de modo: velocidad en acimut
OnMdStby	Cambio de modo: StandBy en acimut
OnMdPoint	Cambio de modo: Posición en acimut
OnMdSlew	Cambio de modo: Velocidad en

	acimut
OnMdStby	Cambio de modo: StandBy en acimut
SetComMode	Actualiza el modo en la estructura de interfaz gráfico
OnEmrgStop	Cambio de modo: Parada de emergencia
SendMsg	Envío de mensajes a través de TheUcom.
OnMdPerr	Solicitud de cambio de modo en elevación: búsqueda

```
////////////////////////////////////////////////////////////////////////////////
// Función: OnSlew
// Descripción: Cambio de modo: velocidad en acimut
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnSlew()
{
        Mode=M_SLEW;
        TrnsPDO=1;
}


////////////////////////////////////////////////////////////////////////////////
// Función: OnMdStby
// Descripción: Cambio de modo: StandBy en acimut
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnStby()
{
        m_SlewCom=0;
        Mode=M_STBY;
        TrnsPDO=1;
}



////////////////////////////////////////////////////////////////////////////////
// Función: OnMdPoint
// Descripción: Cambio de modo: Posición en acimut
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnMdPoint()
```

```
{
	// TODO: Add your control notification handler code here
		Mode=M_POINT;
		if(Mode2==M_SRCH)
		{
			Mode2=M_STBY;
			TrnsPDO=3;
		}
		else
		TrnsPDO=1;
}


////////////////////////////////////////////////////////////////////////////////
// Función: OnMdSlew
// Descripción: Cambio de modo: Velocidad en acimut
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnMdSlew()
{
	// TODO: Add your control notification handler code here
	Mode=M_SLEW;
		if(Mode2==M_SRCH)
		{
			Mode2=M_STBY;
			TrnsPDO=3;
		}
		else
	TrnsPDO=1;
}


////////////////////////////////////////////////////////////////////////////////
// Función: OnMdStby
// Descripción: Cambio de modo: StandBy en acimut
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnMdStby()
{
		Mode=M_STBY;
		if(Mode2==M_SRCH)
		{
			Mode=Mode2=M_STBY;
```

```
                    TrnsPDO=3;
                }
                else
                TrnsPDO=1;
}


////////////////////////////////////////////////////////////////////////////////
// Función: SetComMode
// Descripción: Actualiza el modo en la estructura de interfaz gráfico
// Argumentos: Indice del eje (Az-0 o El-1)
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  SetComMode(int ik)
{
        if(ik==0)
        {
                    if ( InterfazGraficaAz->ComandosMdVelocidad )
                    Mode=M_SLEW;
                    if ( InterfazGraficaAz->ComandosMdPunto )
                    Mode=M_POINT;
                    if ( InterfazGraficaAz->ComandosMdStandBy )
                    Mode=M_STBY;
        }
        else
        {
                    if ( InterfazGraficaEl->ComandosMdVelocidad )
                    Mode2=M_SLEW;
                    if ( InterfazGraficaEl->ComandosMdPunto )
                    Mode2=M_POINT;
                    if ( InterfazGraficaEl->ComandosMdStandBy )
                    Mode2=M_STBY;
        }
}


////////////////////////////////////////////////////////////////////////////////
// Función: OnEmrgStop
// Descripción: Cambio de modo: Parada de emergencia
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnEmrgStop()
{
        // TODO: Add your control notification handler code here
```

```
                Mode=M_EMR_STOP;
                TrnsPDO=3;
                Mode2=M_EMR_STOP;
                TrnsPDO=3;

}



////////////////////////////////////////////////////////////////////////////////
// Función: SendMsg
// Descripción: Envío de mensajes a través de TheUcom. Sólo encapsula la clase, apenas usada
// Parámetros: Buffer de envío, longitud de los datos
////////////////////////////////////////////////////////////////////////////////
void  SendMsg(char BuffOut[], int len)
{
                int len1;
                len1= Ocp.CompMsg(0,BuffOut,len);
                theUCom.Write(BuffOut, len1);

}



////////////////////////////////////////////////////////////////////////////////
// Función: pow
// Descripción: Solicitud de cambio de modo en acimut: búsqueda
// Argumentos:
// Devuelve:
////////////////////////////////////////////////////////////////////////////////
void  OnMdSrch()
{
                Mode=M_SRCH;
                LockCnt=0;
                Hstate=0;
                Mode2=M_SRCH;
                TrnsPDO=3;
}



////////////////////////////////////////////////////////////////////////////////
// Función: OnMdPerr
// Descripción: Solicitud de cambio de modo en elevación: búsqueda
// Argumentos:
// Devuelve:
```

```
////////////////////////////////////////////////////////////////////////////////
void  OnMdSrch2()
{
                Mode=M_SRCH;
                LockCnt=0;
                Hstate=0;
                Mode2=M_SRCH;
                TrnsPDO=3;
}
```