Universidad de Oviedo

Anexos del Trabajo Fin de Máster realizado por

OSCAR ANDRÉS BRAÑA

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Diseño y programación del sistema de control de una turbina mediante programación orientada a objetos

Julio 2016

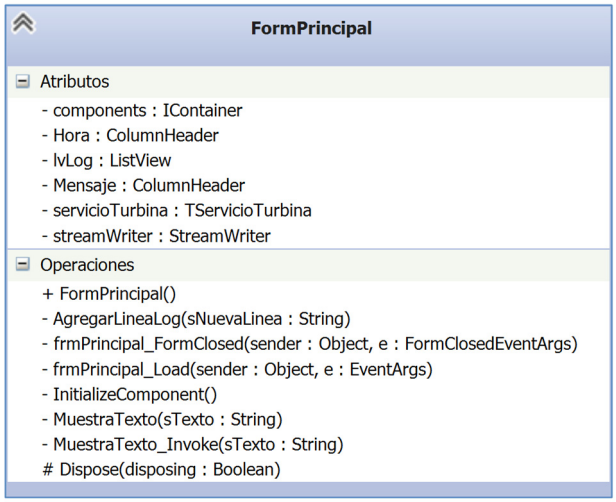
Tutor académico
Felipe Mateos Martín
Tutor de empresa
Jose Luis Pérez Quintas
Empresa
ITRESA – Ingeniería e Informática Industrial de Asturias S.L.

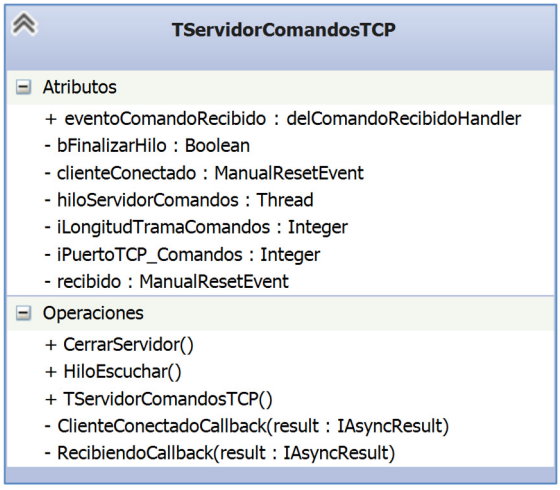
ÍNDICE DE ANEXOS

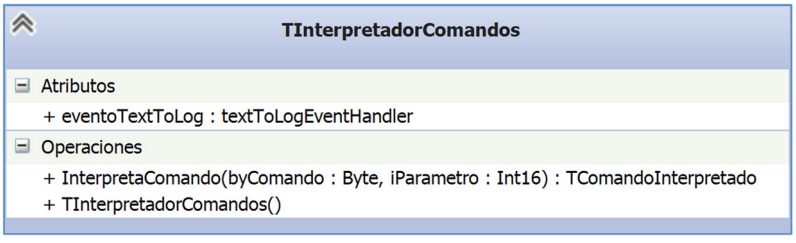
ANEXOS

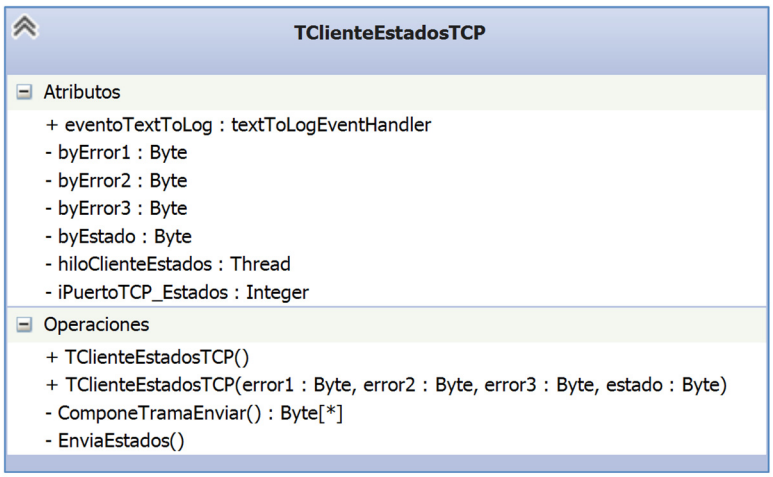
1 ANEXO I: CLASES DEL MODELADO UML

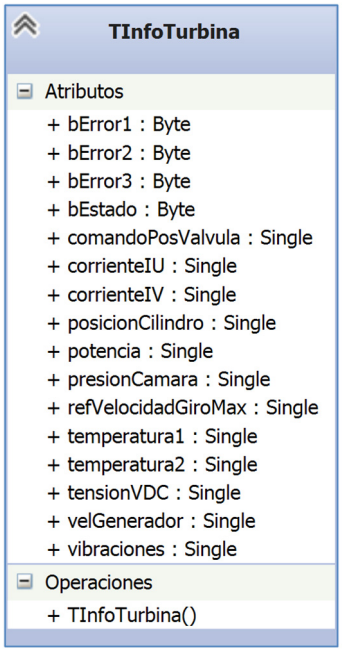
1.1 Aplicación DSP

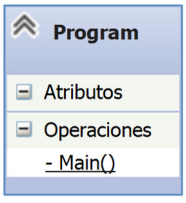


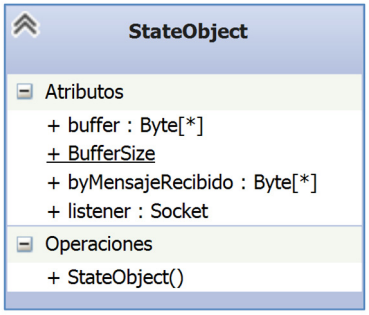


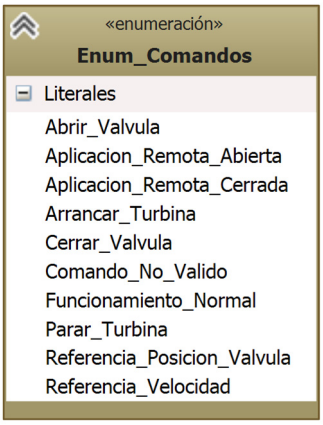


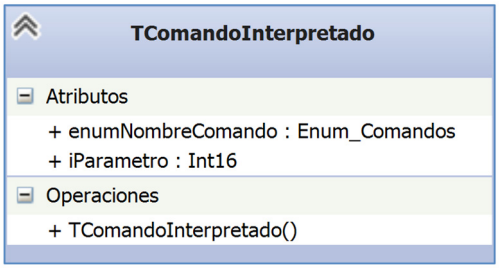

TClienteEstadosTCP
Atributos
+ eventoTextToLog : textToLogEventHandler
- byError1 : Byte
- byError2 : Byte
- byError3 : Byte
- byEstado : Byte
- hiloClienteEstados : Thread
- iPuertoTCP_Estados : Integer
Operaciones
+ TClienteEstadosTCP()
+ TClienteEstadosTCP(error1 : Byte, error2 : Byte, error3 : Byte, estado : Byte)
- ComponeTramaEnviar() : Byte[*]
- EnviaEstados()

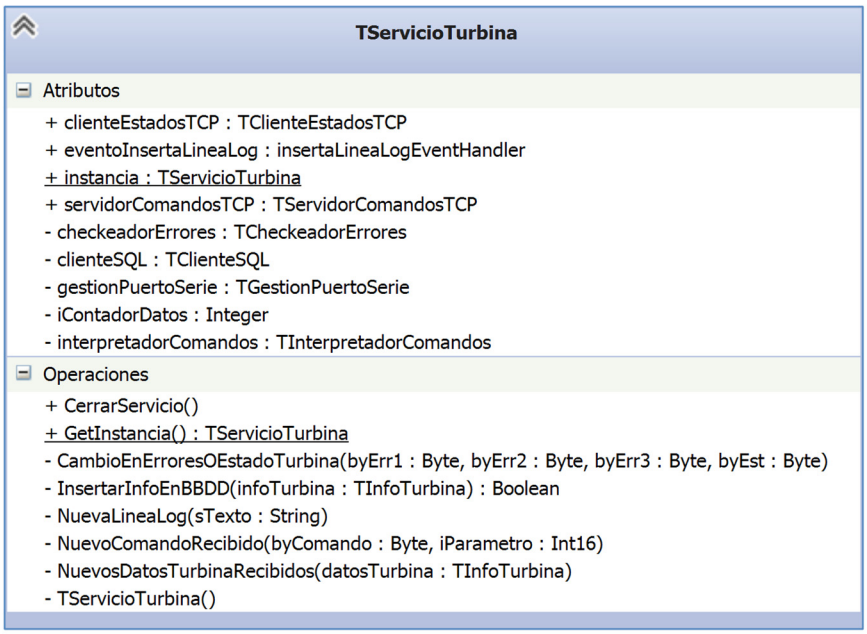

TInfoTurbina
Atributos
+ bError1 : Byte
+ bError2 : Byte
+ bError3 : Byte
+ bEstado : Byte
+ comandoPosValvula : Single
+ corrienteIU : Single
+ corrienteIV : Single
+ posicionCilindro : Single
+ potencia : Single
+ presionCamara : Single
+ refVelocidadGiroMax : Single
+ temperatura1 : Single
+ temperatura2 : Single
+ tensionVDC : Single
+ velGenerador : Single
+ vibraciones : Single
Operaciones
+ TInfoTurbina()

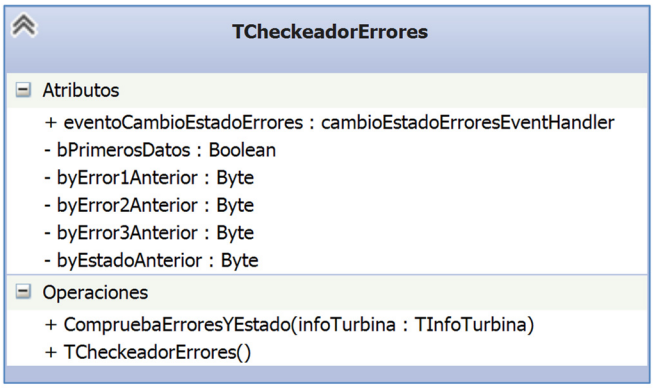

Program
Atributos
Operaciones
- Main()

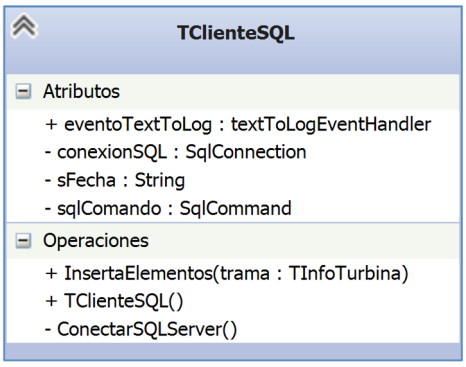

StateObject
Atributos
+ buffer : Byte[*]
+ BufferSize
+ byMensajeRecibido : Byte[*]
+ listener : Socket
Operaciones
+ StateObject()

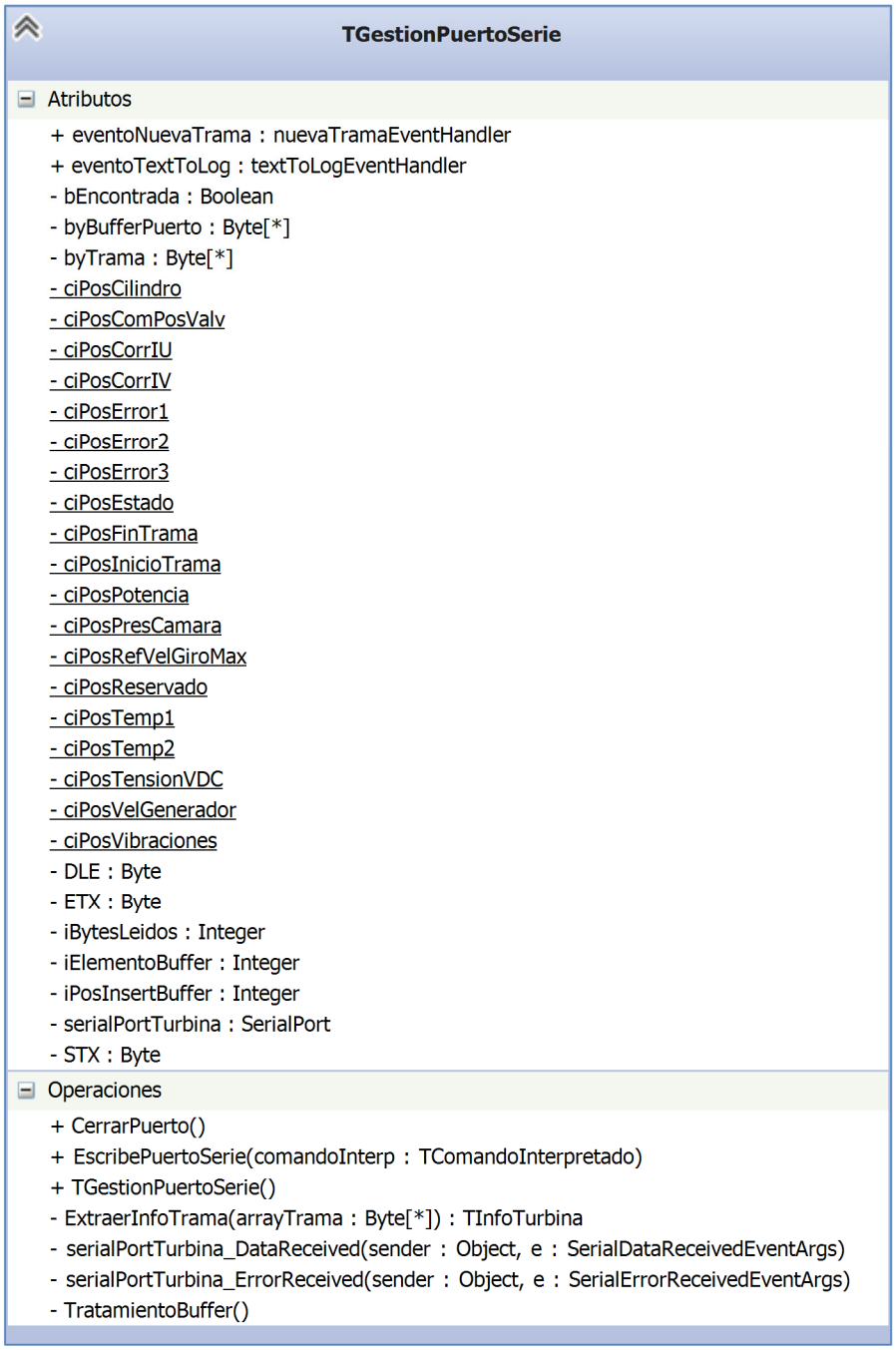

«enumeración»
Enum_Comandos
Literales
Abrir_Valvula
Aplicacion_Remota_Abierta
Aplicacion_Remota_Cerrada
Arrancar_Turbina
Cerrar_Valvula
Comando_No_Valido
Funcionamiento_Normal
Parar_Turbina
Referencia_Posicion_Valvula
Referencia_Velocidad


TComandoInterpretado
Atributos
+ enumNombreComando : Enum_Comandos
+ iParametro : Int16
Operaciones
+ TComandoInterpretado()

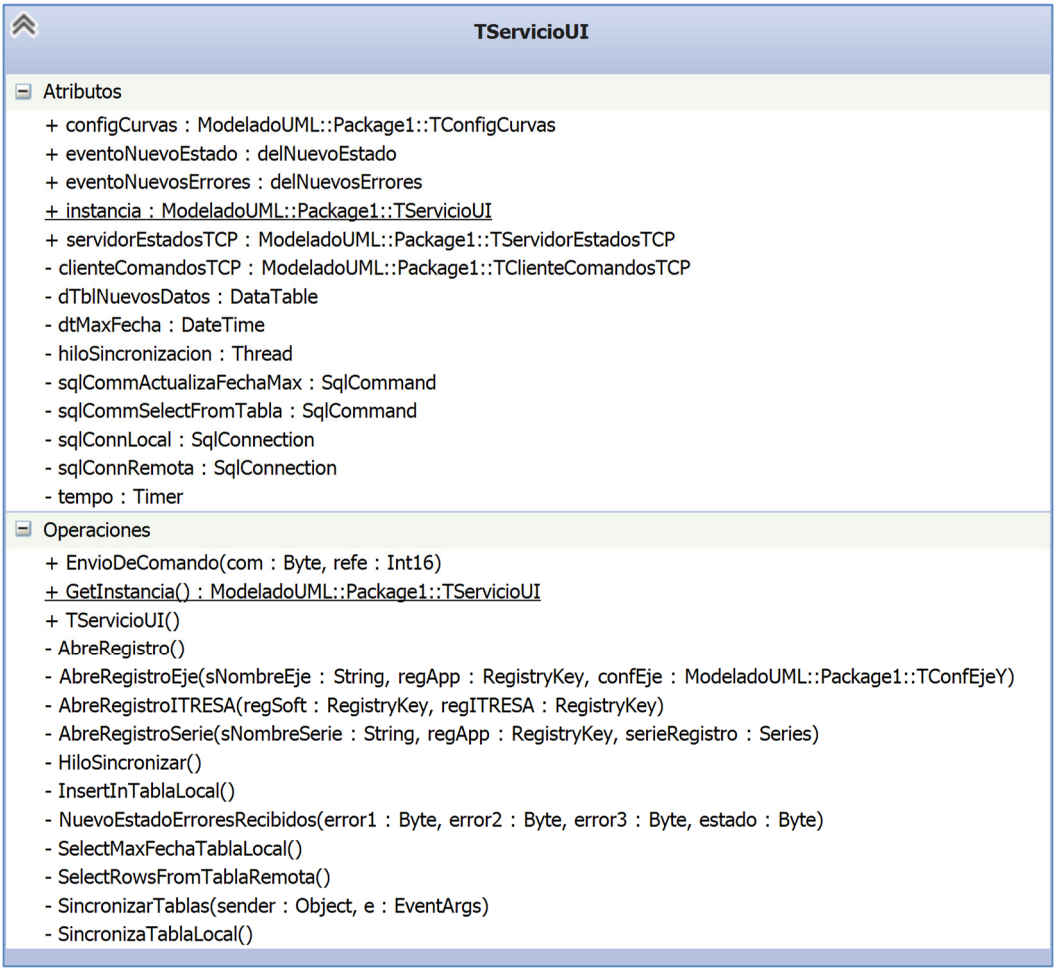

TServicioTurbina
Atributos
+ clienteEstadosTCP : TClienteEstadosTCP
+ eventoInsertaLineaLog : insertaLineaLogEventHandler
+ instancia : TServicioTurbina
+ servidorComandosTCP : TServidorComandosTCP
- checkeadorErrores : TCheckeadorErrores
- clienteSQL : TClienteSQL
- gestionPuertoSerie : TGestionPuertoSerie
- iContadorDatos : Integer
- interpretadorComandos : TInterpretadorComandos
Operaciones
+ CerrarServicio()
+ GetInstancia() : TServicioTurbina
- CambioEnErroresOEstadoTurbina(byErr1 : Byte, byErr2 : Byte, byErr3 : Byte, byEst : Byte)
- InsertarInfoEnBBDD(infoTurbina : TInfoTurbina) : Boolean
- NuevaLineaLog(sTexto : String)
- NuevoComandoRecibido(byComando : Byte, iParametro : Int16)
- NuevosDatosTurbinaRecibidos(datosTurbina : TInfoTurbina)
- TServicioTurbina()

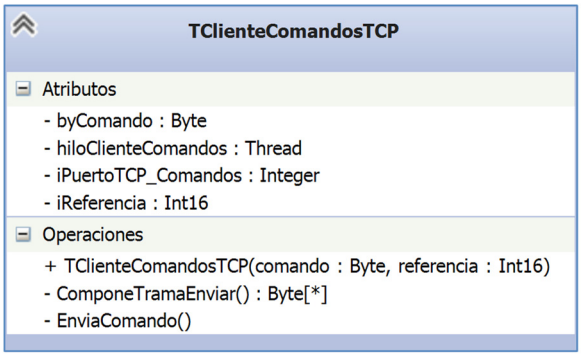

TCheckeadorErrores
Atributos
+ eventoCambioEstadoErrores : cambioEstadoErroresEventHandler
- bPrimerosDatos : Boolean
- byError1Anterior : Byte
- byError2Anterior : Byte
- byError3Anterior : Byte
- byEstadoAnterior : Byte
Operaciones
+ CompruebaErroresYEstado(infoTurbina : TInfoTurbina)
+ TCheckeadorErrores()

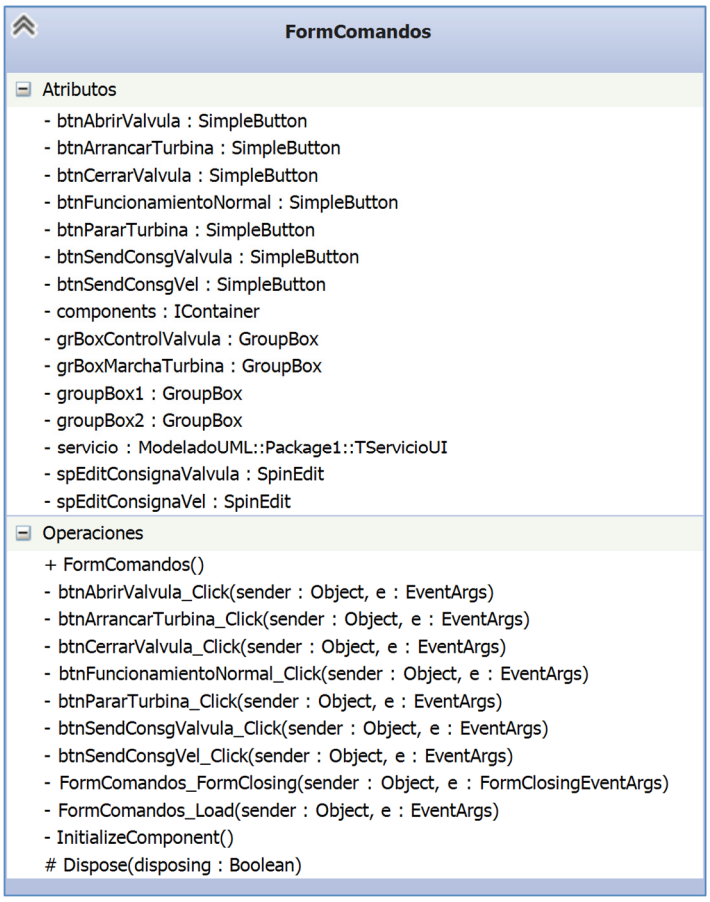

TClienteSQL
Atributos
+ eventoTextToLog : textToLogEventHandler
- conexionSQL : SqlConnection
- sFecha : String
- sqlComando : SqlCommand
Operaciones
+ InsertaElementos(trama : TInfoTurbina)
+ TClienteSQL()
- ConectarSQLServer()

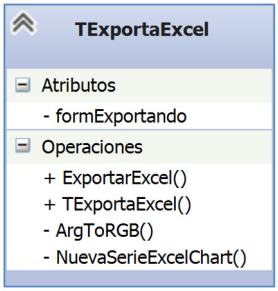

TGestionPuertoSerie
Atributos
+ eventoNuevaTrama : nuevaTramaEventHandler
+ eventoTextToLog : textToLogEventHandler
- bEncontrada : Boolean
- byBufferPuerto : Byte[*]
- byTrama : Byte[*]
- ciPosCilindro
- ciPosComPosValv
- ciPosCorrIU
- ciPosCorrIV
- ciPosError1
- ciPosError2
- ciPosError3
- ciPosEstado
- ciPosFinTrama
- ciPosInicioTrama
- ciPosPotencia
- ciPosPresCamara
- ciPosRefVelGiroMax
- ciPosReservado
- ciPosTemp1
- ciPosTemp2
- ciPosTensionVDC
- ciPosVelGenerador
- ciPosVibraciones
- DLE : Byte
- ETX : Byte
- iBytesLeidos : Integer
- iElementoBuffer : Integer
- iPosInsertBuffer : Integer
- serialPortTurbina : SerialPort
- STX : Byte
Operaciones
+ CerrarPuerto()
+ EscribePuertoSerie(comandoInterp : TComandoInterpretado)
+ TGestionPuertoSerie()
- ExtraerInfoTrama(arrayTrama : Byte[*]) : TInfoTurbina
- serialPortTurbina_DataReceived(sender : Object, e : SerialDataReceivedEventArgs)
- serialPortTurbina_ErrorReceived(sender : Object, e : SerialErrorReceivedEventArgs)
- TratamientoBuffer()

1.2 Aplicación HMI

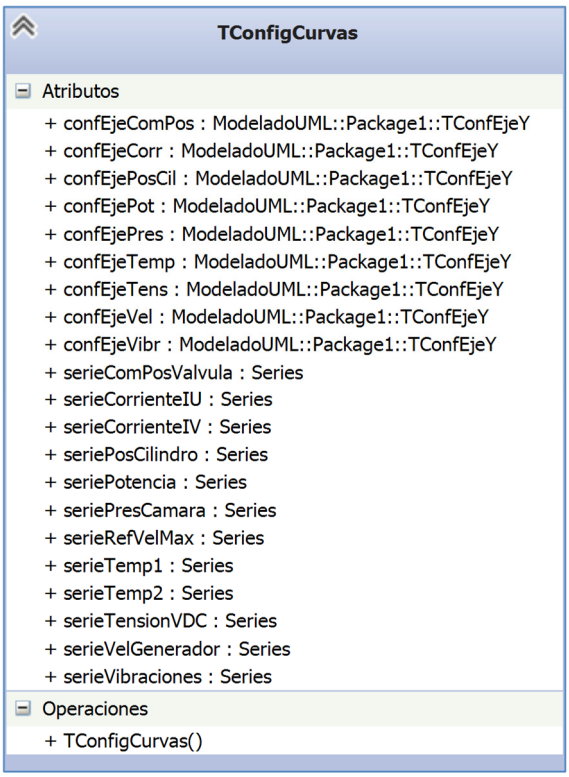


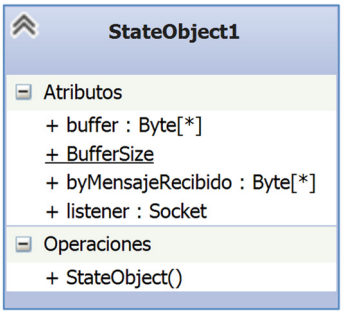


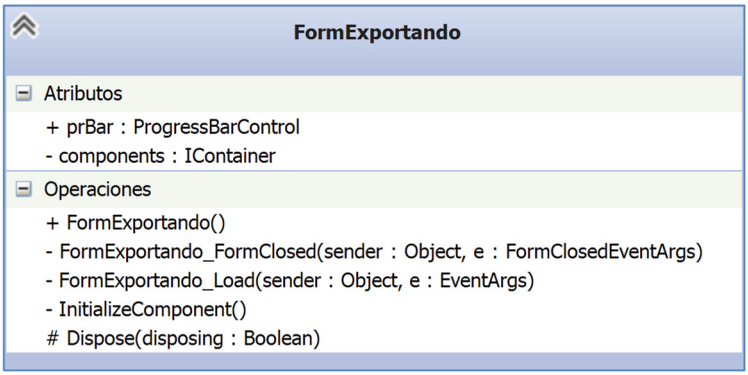

FormComandos
Atributos
- btnAbrirValvula : SimpleButton
- btnArrancarTurbina : SimpleButton
- btnCerrarValvula : SimpleButton
- btnFuncionamientoNormal : SimpleButton
- btnParar Turbina : SimpleButton
- btnSendConsgValvula : SimpleButton
- btnSendConsgVel : SimpleButton
- components : IContainer
- grBoxControlValvula : GroupBox
- grBoxMarchaTurbina : GroupBox
- groupBox1 : GroupBox
- groupBox2 : GroupBox
- servicio : ModeladoUML::Package1::TServicioUI
- spEditConsignaValvula : SpinEdit
- spEditConsignaVel : SpinEdit
Operaciones
+ FormComandos()
- btnAbrirValvula_Click(sender : Object, e : EventArgs)
- btnArrancarTurbina_Click(sender : Object, e : EventArgs)
- btnCerrarValvula_Click(sender : Object, e : EventArgs)
- btnFuncionamientoNormal_Click(sender : Object, e : EventArgs)
- btnParar Turbina_Click(sender : Object, e : EventArgs)
- btnSendConsgValvula_Click(sender : Object, e : EventArgs)
- btnSendConsgVel_Click(sender : Object, e : EventArgs)
- FormComandos_FormClosing(sender : Object, e : FormClosingEventArgs)
- FormComandos_Load(sender : Object, e : EventArgs)
- InitializeComponent()
Dispose(disposing : Boolean)

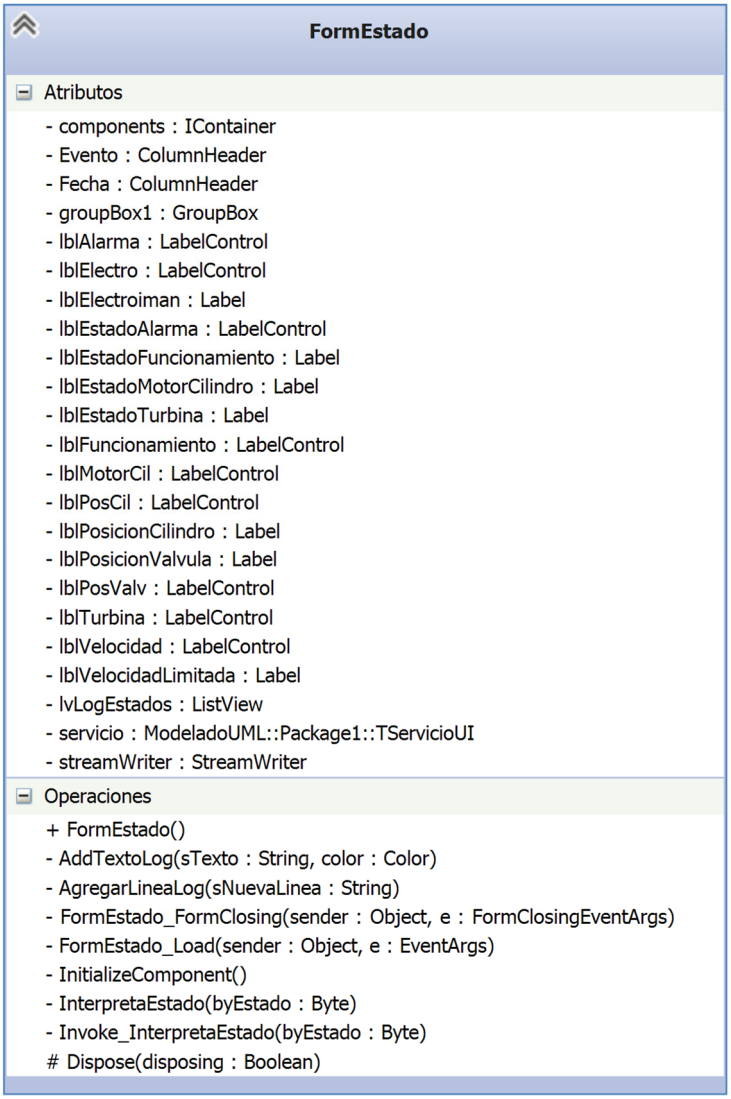

TExportaExcel
Atributos
- formExportando
Operaciones
+ ExportarExcel()
+ TExportaExcel()
- ArgToRGB()
- NuevaSerieExcelChart()

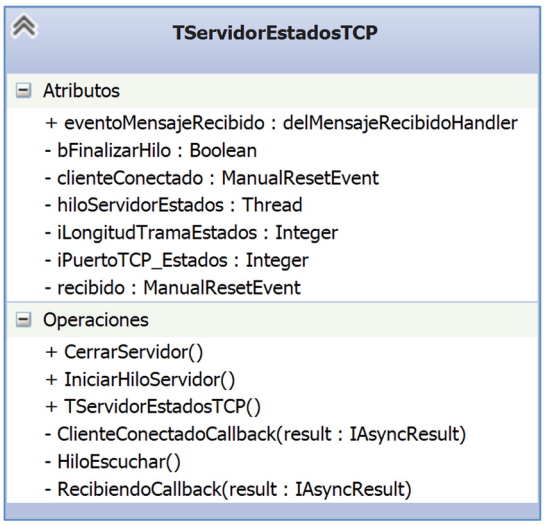

TConfEjeY
Atributos
+ maximoY : Single
+ minimoY : Single
Operaciones
+ TConfEjeY()

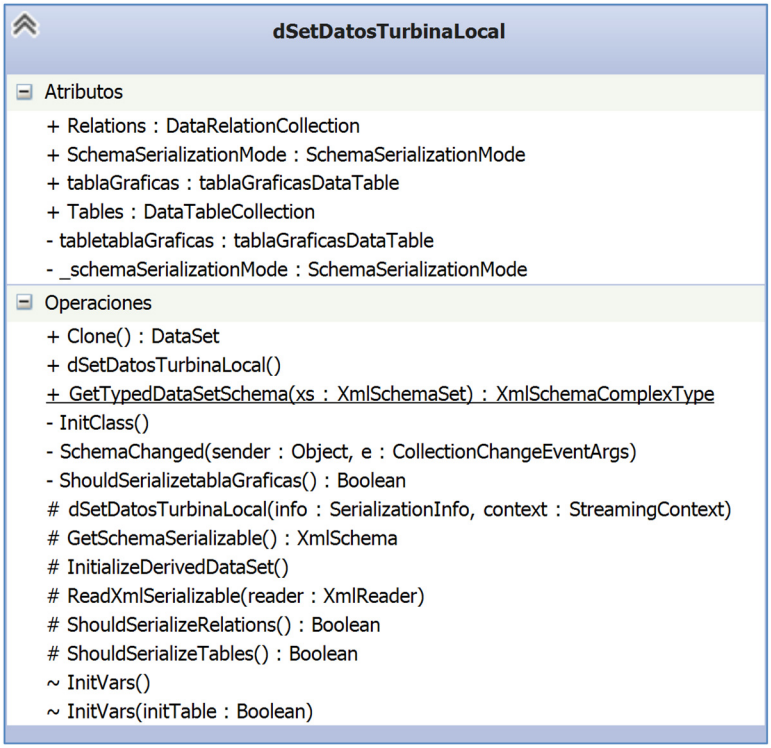

TConfigCurvas
Atributos
+ confEjeComPos : ModeladoUML::Package1::TConfEjeY
+ confEjeCorr : ModeladoUML::Package1::TConfEjeY
+ confEjePosCil : ModeladoUML::Package1::TConfEjeY
+ confEjePot : ModeladoUML::Package1::TConfEjeY
+ confEjePres : ModeladoUML::Package1::TConfEjeY
+ confEjeTemp : ModeladoUML::Package1::TConfEjeY
+ confEjeTens : ModeladoUML::Package1::TConfEjeY
+ confEjeVel : ModeladoUML::Package1::TConfEjeY
+ confEjeVibr : ModeladoUML::Package1::TConfEjeY
+ serieComPosValvula : Series
+ serieCorrienteIU : Series
+ serieCorrienteIV : Series
+ seriePosCilindro : Series
+ seriePotencia : Series
+ seriePresCamara : Series
+ serieRefVelMax : Series
+ serieTemp1 : Series
+ serieTemp2 : Series
+ serieTensionVDC : Series
+ serieVelGenerador : Series
+ serieVibraciones : Series
Operaciones
+ TConfigCurvas()

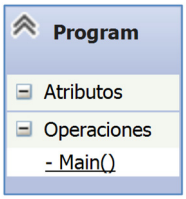

StateObject1
Atributos
+ buffer : Byte[*]
+ BufferSize
+ byMensajeRecibido : Byte[*]
+ listener : Socket
Operaciones
+ StateObject()

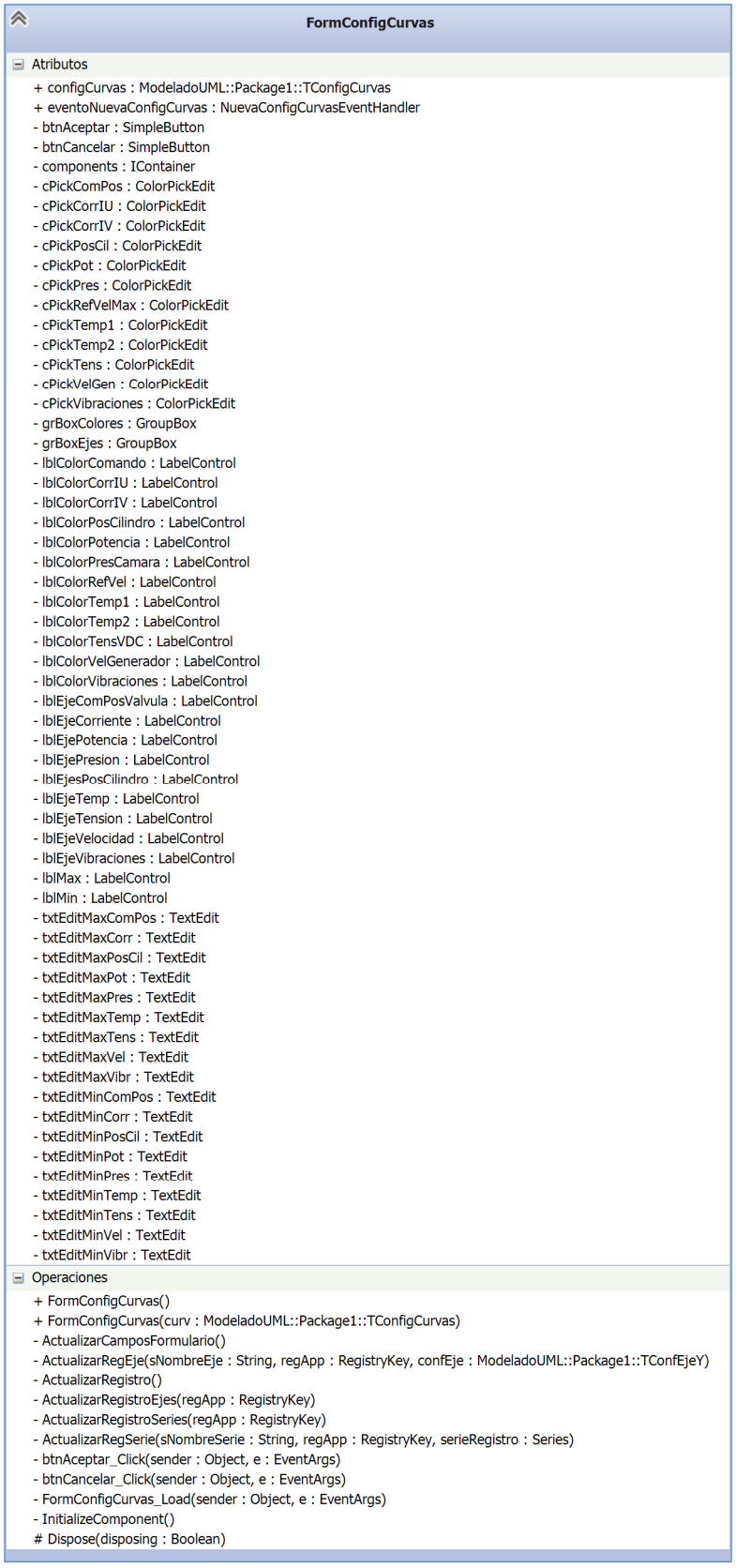

FormExportando
Atributos
+ prBar : ProgressBarControl
- components : IContainer
Operaciones
+ FormExportando()
- FormExportando_FormClosed(sender : Object, e : FormClosedEventArgs)
- FormExportando_Load(sender : Object, e : EventArgs)
- InitializeComponent()
Dispose(disposing : Boolean)

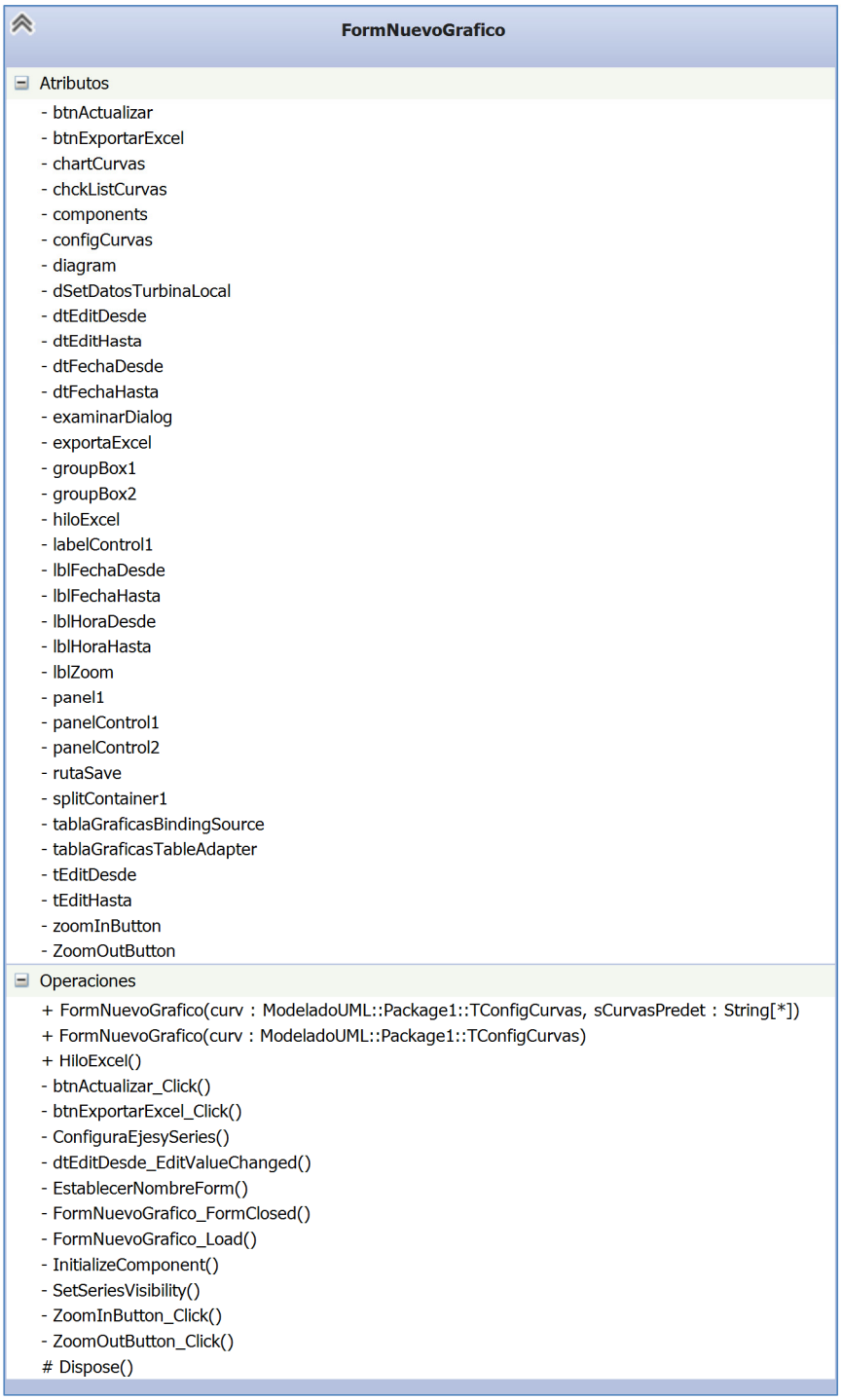

FormEstado
Atributos
- components : IContainer
- Evento : ColumnHeader
- Fecha : ColumnHeader
- groupBox1 : GroupBox
- lblAlarma : LabelControl
- lblElectro : LabelControl
- lblElectroiman : Label
- lblEstadoAlarma : LabelControl
- lblEstadoFuncionamiento : Label
- lblEstadoMotorCilindro : Label
- lblEstadoTurbina : Label
- lblFuncionamiento : LabelControl
- lblMotorCil : LabelControl
- lblPosCil : LabelControl
- lblPosicionCilindro : Label
- lblPosicionValvula : Label
- lblPosValv : LabelControl
- lblTurbina : LabelControl
- lblVelocidad : LabelControl
- lblVelocidadLimitada : Label
- lvLogEstados : ListView
- servicio : ModeladoUML::Package1::TServicioUI
- streamWriter : StreamWriter
Operaciones
+ FormEstado()
- AddTextoLog(sTexto : String, color : Color)
- AgregarLineaLog(sNuevaLinea : String)
- FormEstado_FormClosing(sender : Object, e : FormClosingEventArgs)
- FormEstado_Load(sender : Object, e : EventArgs)
- InitializeComponent()
- InterpretaEstado(byEstado : Byte)
- Invoke_InterpretaEstado(byEstado : Byte)
Dispose(disposing : Boolean)

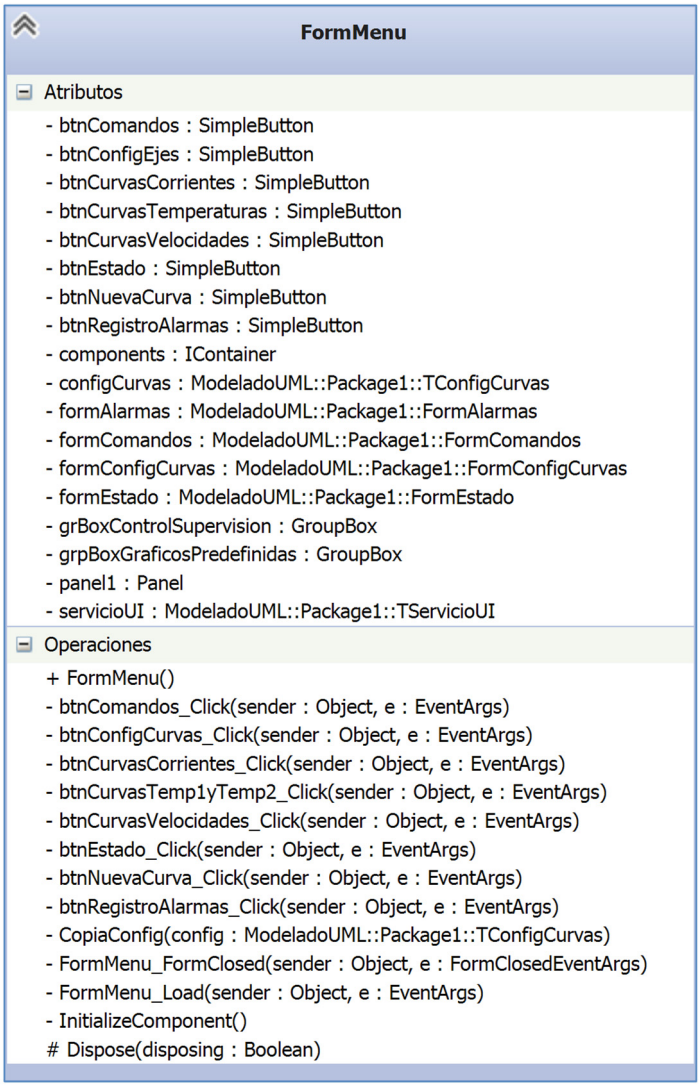

TServidorEstadosTCP
Atributos
+ eventoMensajeRecibido : delMensajeRecibidoHandler
- bFinalizarHilo : Boolean
- clienteConectado : ManualResetEvent
- hiloServidorEstados : Thread
- iLongitudTramaEstados : Integer
- iPuertoTCP_Estados : Integer
- recibido : ManualResetEvent
Operaciones
+ CerrarServidor()
+ IniciarHiloServidor()
+ TServidorEstadosTCP()
- ClienteConectadoCallback(result : IAsyncResult)
- HiloEscuchar()
- RecibiendoCallback(result : IAsyncResult)

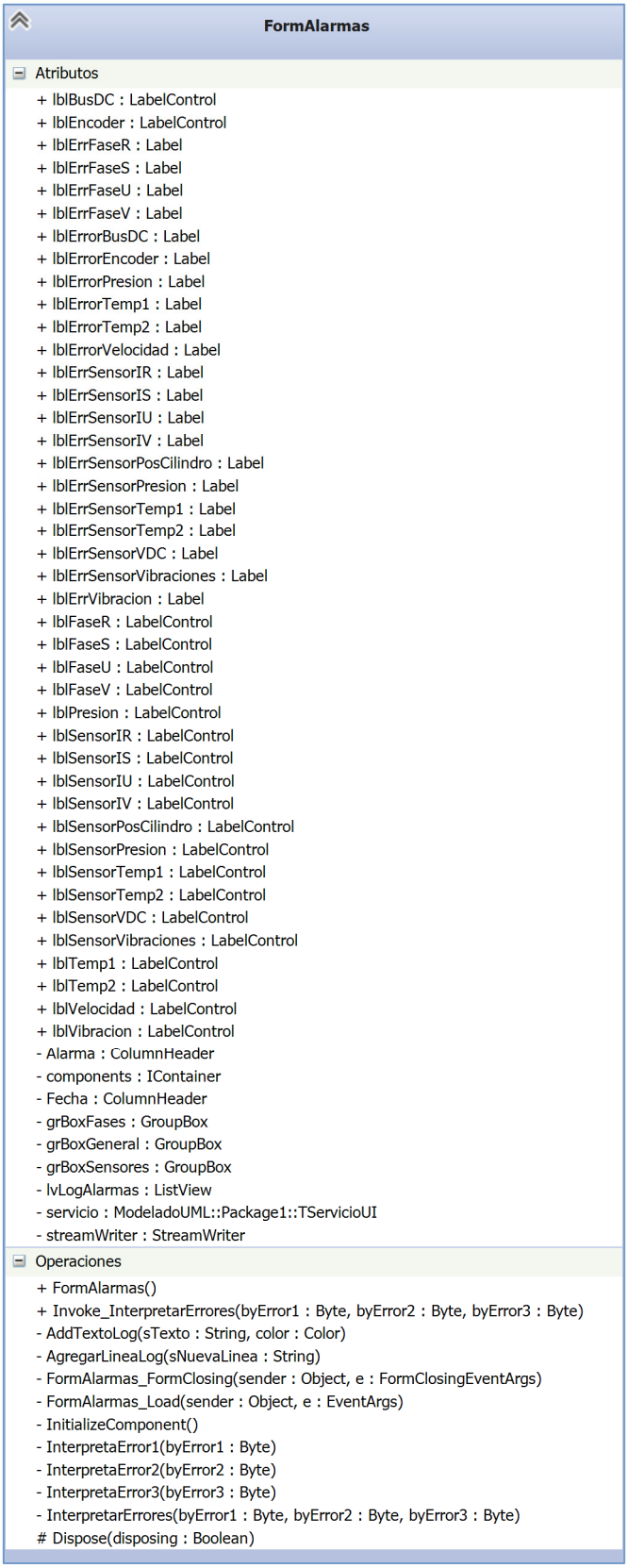

dSetDatosTurbinaLocal
Atributos
+ Relations : DataRelationCollection
+ SchemaSerializationMode : SchemaSerializationMode
+ tablaGraficas : tablaGraficasDataTable
+ Tables : DataTableCollection
- tabletablaGraficas : tablaGraficasDataTable
- _schemaSerializationMode : SchemaSerializationMode
Operaciones
+ Clone() : DataSet
+ dSetDatosTurbinaLocal()
+ GetTypedDataSetSchema(xs : XmlSchemaSet) : XmlSchemaComplexType
- InitClass()
- SchemaChanged(sender : Object, e : CollectionChangeEventArgs)
- ShouldSerializetablaGraficas() : Boolean
dSetDatosTurbinaLocal(info : SerializationInfo, context : StreamingContext)
GetSchemaSerializable() : XmlSchema
InitializeDerivedDataSet()
ReadXmlSerializable(reader : XmlReader)
ShouldSerializeRelations() : Boolean
ShouldSerializeTables() : Boolean
~ InitVars()
~ InitVars(initTable : Boolean)


Program
Atributos
Operaciones
- Main()


FormConfigCurvas
Atributos
+ configCurvas : ModeladoUML::Package1::TConfigCurvas
+ eventoNuevaConfigCurvas : NuevaConfigCurvasEventHandler
- btnAceptar : SimpleButton
- btnCancelar : SimpleButton
- components : IContainer
- cPickComPos : ColorPickEdit
- cPickCorrIU : ColorPickEdit
- cPickCorrIV : ColorPickEdit
- cPickPosCil : ColorPickEdit
- cPickPot : ColorPickEdit
- cPickPres : ColorPickEdit
- cPickRefVelMax : ColorPickEdit
- cPickTemp1 : ColorPickEdit
- cPickTemp2 : ColorPickEdit
- cPickTens : ColorPickEdit
- cPickVelGen : ColorPickEdit
- cPickVibraciones : ColorPickEdit
- grBoxColores : GroupBox
- grBoxEjes : GroupBox
- lblColorComando : LabelControl
- lblColorCorrIU : LabelControl
- lblColorCorrIV : LabelControl
- lblColorPosCilindro : LabelControl
- lblColorPotencia : LabelControl
- lblColorPresCamara : LabelControl
- lblColorRefVel : LabelControl
- lblColorTemp1 : LabelControl
- lblColorTemp2 : LabelControl
- lblColorTensVDC : LabelControl
- lblColorVelGenerador : LabelControl
- lblColorVibraciones : LabelControl
- lblEjeComPosValvula : LabelControl
- lblEjeCorriente : LabelControl
- lblEjePotencia : LabelControl
- lblEjePresion : LabelControl
- lblEjesPosCilindro : LabelControl
- lblEjeTemp : LabelControl
- lblEjeTension : LabelControl
- lblEjeVelocidad : LabelControl
- lblEjeVibraciones : LabelControl
- lblMax : LabelControl
- lblMin : LabelControl
- txtEditMaxComPos : TextEdit
- txtEditMaxCorr : TextEdit
- txtEditMaxPosCil : TextEdit
- txtEditMaxPot : TextEdit
- txtEditMaxPres : TextEdit
- txtEditMaxTemp : TextEdit
- txtEditMaxTens : TextEdit
- txtEditMaxVel : TextEdit
- txtEditMaxVibr : TextEdit
- txtEditMinComPos : TextEdit
- txtEditMinCorr : TextEdit
- txtEditMinPosCil : TextEdit
- txtEditMinPot : TextEdit
- txtEditMinPres : TextEdit
- txtEditMinTemp : TextEdit
- txtEditMinTens : TextEdit
- txtEditMinVel : TextEdit
- txtEditMinVibr : TextEdit
Operaciones
+ FormConfigCurvas()
+ FormConfigCurvas(curv : ModeladoUML::Package1::TConfigCurvas)
- ActualizarCamposFormulario()
- ActualizarRegEje(sNombreEje : String, regApp : RegistryKey, confEje : ModeladoUML::Package1::TConfEjeY)
- ActualizarRegistro()
- ActualizarRegistroEjes(regApp : RegistryKey)
- ActualizarRegistroSeries(regApp : RegistryKey)
- ActualizarRegSerie(sNombreSerie : String, regApp : RegistryKey, serieRegistro : Series)
- btnAceptar_Click(sender : Object, e : EventArgs)
- btnCancelar_Click(sender : Object, e : EventArgs)
- FormConfigCurvas_Load(sender : Object, e : EventArgs)
- InitializeComponent()
Dispose(disposing : Boolean)


FormNuevoGrafico
Atributos
- btnActualizar
- btnExportarExcel
- chartCurvas
- chckListCurvas
- components
- configCurvas
- diagram
- dSetDatosTurbinaLocal
- dtEditDesde
- dtEditHasta
- dtFechaDesde
- dtFechaHasta
- examinarDialog
- exportaExcel
- groupBox1
- groupBox2
- hiloExcel
- labelControl1
- lblFechaDesde
- lblFechaHasta
- lblHoraDesde
- lblHoraHasta
- lblZoom
- panel1
- panelControl1
- panelControl2
- rutaSave
- splitContainer1
- tablaGraficasBindingSource
- tablaGraficasTableAdapter
- tEditDesde
- tEditHasta
- zoomInButton
- ZoomOutButton
Operaciones
+ FormNuevoGrafico(curv : ModeladoUML::Package1::TConfigCurvas, sCurvasPredet : String[*])
+ FormNuevoGrafico(curv : ModeladoUML::Package1::TConfigCurvas)
+ HiloExcel()
- btnActualizar_Click()
- btnExportarExcel_Click()
- ConfiguraEjesySeries()
- dtEditDesde_EditValueChanged()
- EstablecerNombreForm()
- FormNuevoGrafico_FormClosed()
- FormNuevoGrafico_Load()
- InitializeComponent()
- SetSeriesVisibility()
- ZoomInButton_Click()
- ZoomOutButton_Click()
Dispose()


FormMenu
Atributos
- btnComandos : SimpleButton
- btnConfigEjes : SimpleButton
- btnCurvasCorrientes : SimpleButton
- btnCurvasTemperaturas : SimpleButton
- btnCurvasVelocidades : SimpleButton
- btnEstado : SimpleButton
- btnNuevaCurva : SimpleButton
- btnRegistroAlarmas : SimpleButton
- components : IContainer
- configCurvas : ModeladoUML::Package1::TConfigCurvas
- formAlarmas : ModeladoUML::Package1::FormAlarmas
- formComandos : ModeladoUML::Package1::FormComandos
- formConfigCurvas : ModeladoUML::Package1::FormConfigCurvas
- formEstado : ModeladoUML::Package1::FormEstado
- grBoxControlSupervision : GroupBox
- grpBoxGraficosPredefinidas : GroupBox
- panel1 : Panel
- servicioUI : ModeladoUML::Package1::TServicioUI
Operaciones
+ FormMenu()
- btnComandos_Click(sender : Object, e : EventArgs)
- btnConfigCurvas_Click(sender : Object, e : EventArgs)
- btnCurvasCorrientes_Click(sender : Object, e : EventArgs)
- btnCurvasTemp1yTemp2_Click(sender : Object, e : EventArgs)
- btnCurvasVelocidades_Click(sender : Object, e : EventArgs)
- btnEstado_Click(sender : Object, e : EventArgs)
- btnNuevaCurva_Click(sender : Object, e : EventArgs)
- btnRegistroAlarmas_Click(sender : Object, e : EventArgs)
- CopiaConfig(config : ModeladoUML::Package1::TConfigCurvas)
- FormMenu_FormClosed(sender : Object, e : FormClosedEventArgs)
- FormMenu_Load(sender : Object, e : EventArgs)
- InitializeComponent()
Dispose(disposing : Boolean)


FormAlarmas
Atributos
+ lblBusDC : LabelControl
+ lblEncoder : LabelControl
+ lblErrFaseR : Label
+ lblErrFaseS : Label
+ lblErrFaseU : Label
+ lblErrFaseV : Label
+ lblErrorBusDC : Label
+ lblErrorEncoder : Label
+ lblErrorPresion : Label
+ lblErrorTemp1 : Label
+ lblErrorTemp2 : Label
+ lblErrorVelocidad : Label
+ lblErrSensorIR : Label
+ lblErrSensorIS : Label
+ lblErrSensorIU : Label
+ lblErrSensorIV : Label
+ lblErrSensorPosCilindro : Label
+ lblErrSensorPresion : Label
+ lblErrSensorTemp1 : Label
+ lblErrSensorTemp2 : Label
+ lblErrSensorVDC : Label
+ lblErrSensorVibraciones : Label
+ lblErrVibracion : Label
+ lblFaseR : LabelControl
+ lblFaseS : LabelControl
+ lblFaseU : LabelControl
+ lblFaseV : LabelControl
+ lblPresion : LabelControl
+ lblSensorIR : LabelControl
+ lblSensorIS : LabelControl
+ lblSensorIU : LabelControl
+ lblSensorIV : LabelControl
+ lblSensorPosCilindro : LabelControl
+ lblSensorPresion : LabelControl
+ lblSensorTemp1 : LabelControl
+ lblSensorTemp2 : LabelControl
+ lblSensorVDC : LabelControl
+ lblSensorVibraciones : LabelControl
+ lblTemp1 : LabelControl
+ lblTemp2 : LabelControl
+ lblVelocidad : LabelControl
+ lblVibracion : LabelControl
- Alarma : ColumnHeader
- components : IContainer
- Fecha : ColumnHeader
- grBoxFases : GroupBox
- grBoxGeneral : GroupBox
- grBoxSensores : GroupBox
- lvLogAlarmas : ListView
- servicio : ModeladoUML::Package1::TServicioUI
- streamWriter : StreamWriter
Operaciones
+ FormAlarmas()
+ Invoke_InterpretarErrores(byError1 : Byte, byError2 : Byte, byError3 : Byte)
- AddTextoLog(sTexto : String, color : Color)
- AgregarLineaLog(sNuevaLinea : String)
- FormAlarmas_FormClosing(sender : Object, e : FormClosingEventArgs)
- FormAlarmas_Load(sender : Object, e : EventArgs)
- InitializeComponent()
- InterpretaError1(byError1 : Byte)
- InterpretaError2(byError2 : Byte)
- InterpretaError3(byError3 : Byte)
- InterpretarErrores(byError1 : Byte, byError2 : Byte, byError3 : Byte)
Dispose(disposing : Boolean)

2 ANEXO II: CÓDIGO DE LOS PROGRAMAS

2.1 Código aplicación DSP

2.1.1 Program.cs

```
/************************
 ** Program.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ComunicacionTurbina
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormPrincipal());
        }
    }
}
```

2.1.2 FromPrincipal.cs

```
/************************
 ** FormPrincipal.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Threading;
```

```
namespace ComunicacionTurbina
{
    public partial class FormPrincipal : Form
    {
        private TServicioTurbina servicioTurbina;

        private StreamWriter streamWriter;

        public delegate void AddCadenaLog(string sText); //Delegado para mostrar texto en el
log

        #region Métodos
        public FormPrincipal()
        {
            InitializeComponent();

            servicioTurbina = TServicioTurbina.GetInstancia();

            servicioTurbina.eventoInsertaLineaLog += this.MuestraTexto;
        }

        #region Eventos del formulario
        private void frmPrincipal_Load(object sender, EventArgs e)
        {
            this.MuestraTexto("DSP iniciado");
        }

        private void frmPrincipal_FormClosed(object sender, FormClosedEventArgs e)
        {
            servicioTurbina.CerrarServicio();
            this.MuestraTexto("DSP finalizado");
        }
        #endregion

        private void MuestraTexto_Invoke(string sTexto)
        {
            if (lvLog.Items.Count >= 200)
            {//El log muestra 200 elementos como máximo
                lvLog.Items.RemoveAt(199);
            }

            ListViewItem lvItem = new ListViewItem();

            lvItem.Text = DateTime.Now.ToString("dd/mm/yyyy HH:mm:ss.fff");
            lvItem.SubItems.Add(sTexto);
            lvItem.UseItemStyleForSubItems = false;
            lvLog.Items.Insert(0, lvItem); //Inserta elementos en el log siempre en la
primera línea desplazando el resto hacia abajo
            this.AgregarLineaLog(lvItem.Text + "\t|||\t" + lvItem.SubItems[1].Text);
//Agrega una nueva línea al log.txt
        }

        private void MuestraTexto(string sTexto)
        {
```

```
            if (lvLog.InvokeRequired) //Comprueba si otro subproceso distinto al del
formulario requiere al listBox
            {//si se trata de uno distinto
                Invoke(new AddCadenaLog(this.MuestraTexto_Invoke), new object[] { sTexto });
            }
            else
            {//si se trata del proceso del formulario muestra el texto recibido
                MuestraTexto_Invoke(sTexto);
            }
        }

        private void AgregarLineaLog(string sNuevaLinea)
        {
            try
            {
                streamWriter = new StreamWriter(@Properties.Settings.Default.rutaLog, true);
                streamWriter.WriteLine(sNuevaLinea);
                streamWriter.Close();
            }
            catch (Exception ex)
            {
            }
        }
        #endregion
    }

       }
```

2.1.3 TServicioTurbina.cs

```
/***************************
 ** TServicioTurbina.cs **
    31/05/2016
 ***************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;

namespace ComunicacionTurbina
{
    class TServicioTurbina
    {
        private TClienteSQL clienteSQL;
        public TClienteEstadosTCP clienteEstadosTCP;
        private TGestionPuertoSerie gestionPuertoSerie;
        private TInterpretadorComandos interpretadorComandos;
        public TServidorComandosTCP servidorComandosTCP;
        private TCheckeadorErrores checkeadorErrores;
```

```
        public static TServicioTurbina instancia = null;

        public delegate void insertaLineaLogEventHandler(string text);
        public event insertaLineaLogEventHandler eventoInsertaLineaLog;

        private int iContadorDatos = 0;

        private TInfoTurbina infoTurbinaActual = null;

        #region Métodos
        private TServicioTurbina()
        {
            clienteEstadosTCP = new TClienteEstadosTCP();
            clienteSQL = new TClienteSQL();
            gestionPuertoSerie = new TGestionPuertoSerie();
            interpretadorComandos = new TInterpretadorComandos();
            servidorComandosTCP = new TServidorComandosTCP();
            checkeadorErrores = new TCheckeadorErrores();

            gestionPuertoSerie.eventoTextToLog += NuevaLineaLog;
            clienteSQL.eventoTextToLog += NuevaLineaLog;
            interpretadorComandos.eventoTextToLog += NuevaLineaLog;

            gestionPuertoSerie.eventoNuevaTrama += NuevosDatosTurbinaRecibidos;
            servidorComandosTCP.eventoComandoRecibido += NuevoComandoRecibido;
            checkeadorErrores.eventoCambioEstadoErrores += CambioEnErroresOEstadoTurbina;
        }

        private void NuevaLineaLog(string sTexto)
        {
            if(this.eventoInsertaLineaLog!=null)
            {
                this.eventoInsertaLineaLog(sTexto);
            }
        }

        public static TServicioTurbina GetInstancia()
        {
            try
            {
                if (instancia == null)
                {
                    instancia = new TServicioTurbina();
                }
            }
            catch
            {
            }

            return instancia;
        }

        private void CambioEnErroresOEstadoTurbina(byte byErr1, byte byErr2, byte byErr3,
byte byEst)
        {//De checkeadorErrores.eventoCambioEstadoErrores += CambioEnErroresOEstadoTurbina
            //Envío de las palabras de estado y error a la aplicación remota
```

```
            clienteEstadosTCP = new TClienteEstadosTCP(byErr1, byErr2, byErr3, byEst);
            this.eventoInsertaLineaLog("Err1: " + byErr1.ToString() + "; Err2: " +
byErr2.ToString() + "; Err3: " + byErr3.ToString() + "; Est: " + byEst.ToString());
        }

        private void NuevosDatosTurbinaRecibidos(TInfoTurbina datosTurbina)
        {//De gestionPuertoSerie.eventoNuevaTrama += NuevosDatosTurbinaRecibidos
            //Gestión de la información recibida
            if (datosTurbina != null)
            {
                bool bInfoInsertada = this.InsertarInfoEnBBDD(datosTurbina);
                if (bInfoInsertada == true)
                {
                    infoTurbinaActual = datosTurbina;
                    checkeadorErrores.CompruebaErroresYEstado(datosTurbina);

                    iContadorDatos++;

                    if(iContadorDatos%Properties.Settings.Default.iCuentaRecibiendoDatos==0)
                    {
                        this.eventoInsertaLineaLog("Recibiendo datos...");
                        iContadorDatos = 0;
                    }
                }
            }
        }

        private bool InsertarInfoEnBBDD(TInfoTurbina infoTurbina)
        {//Insertar Información recibida en la base de datos local
            bool bRetorno = true;

            try
            {
                clienteSQL.InsertaElementos(infoTurbina);
            }
            catch (System.Data.SqlClient.SqlException ex)
            {
                if (this.eventoInsertaLineaLog != null)
                {
                    this.eventoInsertaLineaLog(ex.Message);
                }

                bRetorno = false;
            }

            return bRetorno;
        }

        private void NuevoComandoRecibido(byte byComando, Int16 iParametro)
        {//De servidorComandosTCP.eventoComandoRecibido += NuevoComandoRecibido
            //gestión del comando recibido de la aplicación remota
            TComandoInterpretado comandoInterpretado =
interpretadorComandos.InterpretaComando(byComando, iParametro);

            gestionPuertoSerie.EscribePuertoSerie(comandoInterpretado);
```

```
            if
((comandoInterpretado.enumNombreComando==Enum_Comandos.Aplicacion_Remota_Abierta) &&
(infoTurbinaActual!=null))
            {
                clienteEstadosTCP = new TClienteEstadosTCP(infoTurbinaActual.bError1,
infoTurbinaActual.bError2, infoTurbinaActual.bError3, infoTurbinaActual.bEstado);
            }
        }

        public void CerrarServicio()
        {
            servidorComandosTCP.CerrarServidor();
            gestionPuertoSerie.CerrarPuerto();
        }
        #endregion
    }
}
```

2.1.4 TServidorComandosTCP.cs

```
/*****************************
 ** TServidorComandosTCP.cs **
    31/05/2016
 *****************************/
using System;
using System.Threading;
using System.Net;
using System.Net.Sockets;

namespace ComunicacionTurbina
{
    class StateObject
    {
        public Socket listener = null;
        public const int BufferSize = 1024;
        public byte[] buffer = new byte[BufferSize];
        public byte[] byMensajeRecibido;
        public StateObject() { }
    }

    class TServidorComandosTCP
    {//Escucha asincrónica de conexiones entrantes vía TCP/IP
        private ManualResetEvent clienteConectado;
        private ManualResetEvent recibido;

        private Thread hiloServidorComandos;

        private int iPuertoTCP_Comandos; //número del puerto de escucha
        private int iLongitudTramaComandos;

        private bool bFinalizarHilo;
```

```
        public delegate void delComandoRecibidoHandler(byte bComando, Int16 iParametro);
        public event delComandoRecibidoHandler eventoComandoRecibido;

        #region Métodos
        public TServidorComandosTCP()
        {
            clienteConectado = new ManualResetEvent(false);
            recibido = new ManualResetEvent(false);
            iPuertoTCP_Comandos = Properties.Settings.Default.iPuertoTCP_Comandos;
            iLongitudTramaComandos = Properties.Settings.Default.iLongTramaComandos;
            bFinalizarHilo = false;

            hiloServidorComandos = new Thread(new ThreadStart(HiloEscuchar));
            hiloServidorComandos.Start();
        }

        public void HiloEscuchar()
        {
            IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Any, iPuertoTCP_Comandos);

            try
            {//Se espera que el cliente se conecte y desconecte del servidor con cada envío
de comandos
                using (Socket listener = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp))
                {
                    listener.Bind(localEndPoint);
                    listener.Listen(1);
                    while (bFinalizarHilo == false)
                    {//Espera por conexión del cliente. Si se desconecta vuelve a esperar
por otra conexión
                        clienteConectado.Reset();
                        listener.BeginAccept(new
AsyncCallback(this.ClienteConectadoCallback), listener);
                        clienteConectado.WaitOne(); //bloqueo
                    }
                }
            }
            catch (SocketException)
            {
            }
        }

        private void ClienteConectadoCallback(IAsyncResult result)
        {
            clienteConectado.Set(); //cliente conectado, desbloqueo
            StateObject st = new StateObject();
            try
            {
                st.listener = (Socket)result.AsyncState;
                st.listener = st.listener.EndAccept(result);
                //Espera a que termine la recepción de comandos
                recibido.Reset();
                st.listener.BeginReceive(st.buffer, 0, StateObject.BufferSize,
SocketFlags.None, new AsyncCallback(this.RecibiendoCallback), st);
                recibido.WaitOne(); //bloqueo
```

```
            }
            catch (SocketException)
            {
            }
            catch (ObjectDisposedException)
            { }
        }

        private void RecibiendoCallback(IAsyncResult result)
        {
            StateObject st = (StateObject)result.AsyncState;
            try
            {
                recibido.Set(); //recepción terminada, desbloqueo
                int bytesRecibidos = st.listener.EndReceive(result);
                st.byMensajeRecibido = new byte[bytesRecibidos];

                if (bytesRecibidos == iLongitudTramaComandos)
                {//Extracción de los bytes del comando de la trama completa (de 1024 bytes)
                    Buffer.BlockCopy(st.buffer, 0, st.byMensajeRecibido, 0, bytesRecibidos);

                    if (this.eventoComandoRecibido != null)
                    {//en el byte 2 está el tipo de comando y en el 3 y 4 las partes alta y
baja de la referencia cuando corresponda
                        Int16 iParametro = (Int16)((Int16)st.byMensajeRecibido[3] << 8 |
st.byMensajeRecibido[4]);
                        this.eventoComandoRecibido(st.byMensajeRecibido[2], iParametro);
                    }
                }
            }
            catch (SocketException)
            {
            }
        }

        public void CerrarServidor()
        {//Cierra el servidor
            bFinalizarHilo = true;
            clienteConectado.Set();
        }
        #endregion
    }
}
```

2.1.5 TInterpretadorComandos.cs

```
/*********************************
 ** TInterpretadorComandos.cs **
    31/05/2016
 *********************************/
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ComunicacionTurbina
{
    enum Enum_Comandos
    {//Enumeración de los posibles comandos que se pueden recibir
        Funcionamiento_Normal,
        Parar_Turbina,
        Arrancar_Turbina,
        Abrir_Valvula,
        Cerrar_Valvula,
        Referencia_Velocidad,
        Referencia_Posicion_Valvula,
        Aplicacion_Remota_Abierta,
        Aplicacion_Remota_Cerrada,
        Comando_No_Valido
    }

    class TComandoInterpretado
    {
        public Enum_Comandos enumNombreComando;
        public Int16 iParametro;

        public TComandoInterpretado()
        {
            iParametro = 0;
        }
    }

    class TInterpretadorComandos
    {
        public delegate void textToLogEventHandler(string text);
        public event textToLogEventHandler eventoTextToLog;

        #region Métodos
        public TInterpretadorComandos()
        {

        }

        public TComandoInterpretado InterpretaComando(byte byComando, Int16 iParametro)
        {
            TComandoInterpretado comandoInterpretado = new TComandoInterpretado();

            switch (byComando)
            {
                case 0x00: //Comando funcionamiento normal recibido
                    comandoInterpretado.enumNombreComando =
Enum_Comandos.Funcionamiento_Normal;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Funcionamiento normal\" recibido");
                    }
                    break;
```

```
                case 0xFF: //Comando parar turbina recibido
                    comandoInterpretado.enumNombreComando = Enum_Comandos.Parar_Turbina;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Parar turbina\" recibido");
                    }
                    break;

                case 0xAA: //Comando arrancar turbina recibido
                    comandoInterpretado.enumNombreComando = Enum_Comandos.Arrancar_Turbina;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Arrancar turbina\" recibido");
                    }
                    break;

                case 0xF0: //Comando abrir válvula recibido
                    comandoInterpretado.enumNombreComando = Enum_Comandos.Abrir_Valvula;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Abrir válvula\" recibido");
                    }
                    break;

                case 0x0F: //Comando cerrar válvula recibido
                    comandoInterpretado.enumNombreComando = Enum_Comandos.Cerrar_Valvula;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Cerrar válvula\" recibido");
                    }
                    break;

                case 0x0A: //Comando consigna velocidad máxima recibido
                    comandoInterpretado.enumNombreComando =
Enum_Comandos.Referencia_Velocidad;
                    comandoInterpretado.iParametro = iParametro;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Referencia velocidad giro máx. (" +
iParametro + ")\" recibido");
                    }
                    break;

                case 0xA0: //Comando referencia posición válvula recibido
                    comandoInterpretado.enumNombreComando =
Enum_Comandos.Referencia_Posicion_Valvula;
                    comandoInterpretado.iParametro = iParametro;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando \"Referencia posición válvula (" +
iParametro + ")\" recibido");
                    }
                    break;

                case 0xFA: //Cuando se inicia la aplicación HMI se recibe este comando
```

```
                    comandoInterpretado.enumNombreComando =
Enum_Comandos.Aplicacion_Remota_Abierta;

                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("La interfaz HMI se ha iniciado");
                    }
                    break;

                case 0xAF: //Cuando se cierra la aplicación HMI se recibe este comando
                    comandoInterpretado.enumNombreComando =
Enum_Comandos.Aplicacion_Remota_Cerrada;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("La interfaz HMI se ha cerrado");
                    }
                    break;

                default: //Comando no válido
                    comandoInterpretado.enumNombreComando = Enum_Comandos.Comando_No_Valido;
                    if (this.eventoTextToLog != null)
                    {
                        this.eventoTextToLog("Comando recibido no válido.");
                    }
                    break;
            }

            return comandoInterpretado;
        }
        #endregion
    }
}
```

2.1.6 TInfoTurbina.cs

```
/************************
 ** TInfoTurbina.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ComunicacionTurbina
{
    public class TInfoTurbina
    {
        //Campos de la trama
        public float velGenerador;
        public float presionCamara;
```

```
        public float posicionCilindro;
        public float potencia;
        public float vibraciones;
        public float corrienteIU;
        public float corrienteIV;
        public float tensionVDC;
        public float temperatura1;
        public float temperatura2;
        public float comandoPosValvula;
        public float refVelocidadGiroMax;
        public byte bError1;
        public byte bError2;
        public byte bError3;
        public byte bEstado;

        #region Métodos
        public TInfoTurbina()
        {
        }
        #endregion
    }

    }
```

2.1.7 TGestionPuertoSerie.cs

```
/*****************************
 ** TGestionPuertoSerie.cs **
    31/05/2016
 *****************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.IO;

namespace ComunicacionTurbina
{
    class TGestionPuertoSerie
    {
        #region Constantes del protocolo

        //Índices de inicio de las palabras de dos bytes correspondientes a cada campo
dentro de la trama recibida
        private const int ciPosInicioTrama = 0;
        private const int ciPosVelGenerador = 2;
        private const int ciPosPresCamara = 4;
        private const int ciPosCilindro = 6;
        private const int ciPosPotencia = 8;
        private const int ciPosVibraciones = 10;
```

```
        private const int ciPosCorrIU = 12;
        private const int ciPosCorrIV = 14;
        private const int ciPosTensionVDC = 16;
        private const int ciPosTemp1 = 18;
        private const int ciPosTemp2 = 20;
        private const int ciPosComPosValv = 22;
        private const int ciPosRefVelGiroMax = 24;
        private const int ciPosError1 = 26;
        private const int ciPosError2 = 27;
        private const int ciPosError3 = 28;
        private const int ciPosEstado = 29;
        private const int ciPosReservado = 30;
        private const int ciPosFinTrama = 31;
        //Palabras de control
        private byte DLE = Properties.Settings.Default.DLE;
        private byte STX = Properties.Settings.Default.STX;
        private byte ETX = Properties.Settings.Default.ETX;

        #endregion

        private System.IO.Ports.SerialPort serialPortTurbina;

        private int iBytesLeidos;
        private bool bEncontrada;
        private int iElementoBuffer;
        private int iPosInsertBuffer;
        private byte[] byBufferPuerto; //Búffer de adquisición
        private byte[] byTrama; //Trama extraída del búffer de adquisición

        public delegate void textToLogEventHandler(string text);
        public event textToLogEventHandler eventoTextToLog;

        public delegate void nuevaTramaEventHandler(TInfoTurbina datosTurbina);
        public event nuevaTramaEventHandler eventoNuevaTrama;

        #region Métodos
        public TGestionPuertoSerie()
        {
            serialPortTurbina = new System.IO.Ports.SerialPort();
            //Parámetros de configuración del puerto serie
            serialPortTurbina.ReadBufferSize = Properties.Settings.Default.iLongBuffer;
            serialPortTurbina.BaudRate = Properties.Settings.Default.iPuertoBaudRate;
            serialPortTurbina.DataBits = Properties.Settings.Default.iPuertoDataBits;
            serialPortTurbina.PortName = Properties.Settings.Default.sCOMPuerto;

            this.serialPortTurbina.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(this.serialPortTurbina_DataReceived);

            try
            {
                serialPortTurbina.Open(); //Abrir puerto serie

                serialPortTurbina.DiscardInBuffer(); //Elimina posibles datos anteriores en
el búffer de entrada
                serialPortTurbina.DiscardOutBuffer(); //Elimina posibles datos anteriores en
el búffer de salida
```

```
                bEncontrada = false;
                iPosInsertBuffer = 0;
                byBufferPuerto = new byte[4096];
                byTrama = new byte[Properties.Settings.Default.iLongTrama];
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("Puerto " + serialPortTurbina.PortName.ToString() +
" abierto");
                }
            }
            catch (UnauthorizedAccessException)
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("El puerto seleccionado ya está en uso. Verifique
su estado y reinicie el DSP.");
                }
            }
            catch (ArgumentException)
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("Imposible abrir el puerto seleccionado. No existe
o su nombre no es correcto. Reinicie el DSP.");
                }
            }
            catch (IOException)
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("Imposible abrir el puerto serie. Reinicie el
DSP.");
                }
            }
            catch (InvalidOperationException)
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("El puerto seleccionado ya está abierto. Compruebe
su estado y reinicie el DSP.");
                }
            }
        }

        #region Recepción de trama y tratamiento del búffer
        private void serialPortTurbina_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {//Datos recibidos en el puerto
            try
            {
                iBytesLeidos = serialPortTurbina.BytesToRead; //Bytes leídos

                if (byBufferPuerto != null)
                {
```

```
                    if (iPosInsertBuffer + iBytesLeidos <
Properties.Settings.Default.iLongBuffer - 1)
                    {//Si hay sitio en el búffer para nuevos datos
                        serialPortTurbina.Read(byBufferPuerto, iPosInsertBuffer,
iBytesLeidos); //Lee los datos de la posición correspondiente
                        iPosInsertBuffer += iBytesLeidos; //Actualiza la posición para la
siguiente lectura

                        if (iPosInsertBuffer >= (Properties.Settings.Default.iLongTrama *
2))
                        {//Cogiendo un tamaño dos veces mayor que el tamaño de la trama, se
espera encontrar al menos un comienzo de trama
                            this.TratamientoBuffer();
                        }
                    }
                    else
                    {//Si no hay sitio en el búffer para nuevos datos lo vuelve a rellenar
desde el pricipio
                        iPosInsertBuffer = 0;
                        serialPortTurbina.Read(byBufferPuerto, iPosInsertBuffer,
iBytesLeidos);
                        serialPortTurbina.DiscardInBuffer();
                    }
                }
            }
            catch (Exception ex)
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog(ex.Message);
                }
            }
        }

        private void TratamientoBuffer()
        {
            bEncontrada = false;
            iElementoBuffer = 0;
            while ((bEncontrada == false) && (iElementoBuffer < iPosInsertBuffer -
Properties.Settings.Default.iLongTrama))
            {//Localizar el DLE STX de inicio de la trama y el DLE ETX de fin
                if ((byBufferPuerto[iElementoBuffer] == Properties.Settings.Default.DLE) &&
                    (byBufferPuerto[iElementoBuffer + 1] ==
Properties.Settings.Default.STX))
                {
                    bEncontrada = true;
                }
                else
                {
                    iElementoBuffer++;
                }
            }

            if (bEncontrada == true)
            {//Si se encuentra una trama
```

```
                if ((byBufferPuerto[iElementoBuffer +
(Properties.Settings.Default.iLongTrama - 2)] == Properties.Settings.Default.DLE) &&
                    (byBufferPuerto[iElementoBuffer +
(Properties.Settings.Default.iLongTrama - 1)] == Properties.Settings.Default.ETX))
                {//Si el número de elementos entre el inicio y fin de la trama son el
esperado
                    try
                    {
                        //Copia la trama para extraer la información
                        Buffer.BlockCopy(byBufferPuerto, iElementoBuffer, byTrama, 0,
Properties.Settings.Default.iLongTrama);
                        TInfoTurbina datosTrama = null;

                        //Sobreescribe los elementos del búffer copiados rotando los que no
se han copiado hacia el inicio
                        for (int i = iElementoBuffer +
Properties.Settings.Default.iLongTrama; i < iPosInsertBuffer; i++)
                        {
                            byBufferPuerto[i - iElementoBuffer -
Properties.Settings.Default.iLongTrama] = byBufferPuerto[i];
                        }
                        //Actualiza la nueva posición para insertar en el búffer
                        iPosInsertBuffer = iPosInsertBuffer - (iElementoBuffer +
Properties.Settings.Default.iLongTrama);

                        //Extraer la información contenida en la trama
                        datosTrama = ExtraerInfoTrama(byTrama);

                        if ( (this.eventoNuevaTrama != null) && (datosTrama != null))
                        {
                            this.eventoNuevaTrama(datosTrama); //Informa a servicioTurbina
de que se ha recibido una nueva trama y se la pasa como parámetro
                        }
                    }
                    catch (Exception ex)
                    {
                        if (this.eventoTextToLog != null)
                        {
                            this.eventoTextToLog(ex.Message);
                        }
                    }
                }
            }
            else
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("Trama erronea");
                }
                iPosInsertBuffer = 0;
            }
        }
        #endregion

        public void EscribePuertoSerie(TComandoInterpretado comandoInterp)
        {
```

```
            byte[] bufferEnvioPuertoSerie = new byte[7];
            byte[] bufferReferencia = null;
            bool bComandoOK = true; //Indica que se recibe un comando válido

            bufferEnvioPuertoSerie[0] = Properties.Settings.Default.DLE;
            bufferEnvioPuertoSerie[1] = Properties.Settings.Default.STX;
            bufferEnvioPuertoSerie[2] = 0;
            bufferEnvioPuertoSerie[3] = 0;
            bufferEnvioPuertoSerie[4] = 0;
            bufferEnvioPuertoSerie[5] = Properties.Settings.Default.DLE;
            bufferEnvioPuertoSerie[6] = Properties.Settings.Default.ETX;

            switch (comandoInterp.enumNombreComando)
            {
                case Enum_Comandos.Funcionamiento_Normal: //Comando funcionamiento normal
recibido
                    //Inicialmente bufferenvio[2] ya esta a cero: bufferEnvioPuertoSerie[2]
= 0x00;
                    break;

                case Enum_Comandos.Parar_Turbina: //Comando parar turbina recibido
                    bufferEnvioPuertoSerie[2] = 0xFF;
                    break;

                case Enum_Comandos.Arrancar_Turbina: //Comando arrancar turbina recibido
                    bufferEnvioPuertoSerie[2] = 0xAA;
                    break;

                case Enum_Comandos.Abrir_Valvula: //Comando abrir válvula recibido
                    bufferEnvioPuertoSerie[2] = 0xF0;
                    break;

                case Enum_Comandos.Cerrar_Valvula: //Comando cerrar válvula recibido
                    bufferEnvioPuertoSerie[2] = 0x0F;
                    break;

                case Enum_Comandos.Referencia_Velocidad: //Comando consigna velocidad máxima
recibido
                    bufferEnvioPuertoSerie[2] = 0x0A;
                    try
                    {//Obtiene los bytes alto y bajo de la referencia de velocidad
                        bufferReferencia = BitConverter.GetBytes(comandoInterp.iParametro);
                        bufferEnvioPuertoSerie[3] = bufferReferencia[1];
                        bufferEnvioPuertoSerie[4] = bufferReferencia[0];
                    }
                    catch
                    {
                    }
                    break;

                case Enum_Comandos.Referencia_Posicion_Valvula: //Comando referencia
posición válvula recibido
                    bufferEnvioPuertoSerie[2] = 0xA0;
                    try
                    {//Obtiene los bytes alto y bajo de la referencia de posición de la
válvula
```

```
                        bufferReferencia = BitConverter.GetBytes(comandoInterp.iParametro);
                        bufferEnvioPuertoSerie[3] = bufferReferencia[1];
                        bufferEnvioPuertoSerie[4] = bufferReferencia[0];
                    }
                    catch
                    {
                    }
                    break;

                default:
                    //Ningún comando válido que enviar
                    bComandoOK = false;
                    break;
            }

            if (bComandoOK == true)
            {//Si el comando recibido es válido se envía a la turbina a través del puerto
serie
                try
                {
                    serialPortTurbina.Write(bufferEnvioPuertoSerie, 0,
bufferEnvioPuertoSerie.Length);
                }
                catch(Exception ex)
                {
                    this.eventoTextToLog("Advertencia: no se pueden enviar datos a través
del puerto serie. " + ex.Message);
                }
            }

            try
            {
                bufferEnvioPuertoSerie = null;
                bufferReferencia = null;
            }
            catch
            {
            }
        }

        private void serialPortTurbina_ErrorReceived(object sender,
System.IO.Ports.SerialErrorReceivedEventArgs e)
        {
            if (this.eventoTextToLog != null)
            {
                this.eventoTextToLog(e.EventType.ToString());
            }
        }

        public void CerrarPuerto()
        {
            serialPortTurbina.Close();
        }

        private TInfoTurbina ExtraerInfoTrama(byte[] arrayTrama)
        {
```

```
            TInfoTurbina infoTurbina = new TInfoTurbina();

            try
            {

                if (arrayTrama[ciPosInicioTrama] == DLE && arrayTrama[ciPosInicioTrama + 1]
== STX
                    && arrayTrama[ciPosFinTrama] == DLE && arrayTrama[ciPosFinTrama + 1] ==
ETX)
                {//Extrae la informacion de los campos de la trama teniendo en cuenta que
están formados por dos bytes
                    infoTurbina.velGenerador = (float)((short)arrayTrama[ciPosVelGenerador]
<< 8 | arrayTrama[ciPosVelGenerador + 1]);
                    infoTurbina.presionCamara = (float)((short)arrayTrama[ciPosPresCamara]
<< 8 | arrayTrama[ciPosPresCamara + 1]);
                    infoTurbina.posicionCilindro = (float)((short)arrayTrama[ciPosCilindro]
<< 8 | arrayTrama[ciPosCilindro + 1]);
                    infoTurbina.potencia = (float)((short)arrayTrama[ciPosPotencia] << 8 |
arrayTrama[ciPosPotencia + 1]);
                    infoTurbina.vibraciones = (float)((short)arrayTrama[ciPosVibraciones] <<
8 | arrayTrama[ciPosVibraciones + 1]);
                    infoTurbina.corrienteIU = (float)((short)arrayTrama[ciPosCorrIU] << 8 |
arrayTrama[ciPosCorrIU + 1]);
                    infoTurbina.corrienteIV = (float)((short)arrayTrama[ciPosCorrIV] << 8 |
arrayTrama[ciPosCorrIV + 1]);
                    infoTurbina.tensionVDC = (float)((short)arrayTrama[ciPosTensionVDC] << 8
| arrayTrama[ciPosTensionVDC + 1]);
                    infoTurbina.temperatura1 = (float)((short)arrayTrama[ciPosTemp1] << 8 |
arrayTrama[ciPosTemp1 + 1]);
                    infoTurbina.temperatura2 = (float)((short)arrayTrama[ciPosTemp2] << 8 |
arrayTrama[ciPosTemp2 + 1]);
                    infoTurbina.comandoPosValvula =
(float)((short)arrayTrama[ciPosComPosValv] << 8 | arrayTrama[ciPosComPosValv + 1]);
                    infoTurbina.refVelocidadGiroMax =
(float)((short)arrayTrama[ciPosRefVelGiroMax] << 8 | arrayTrama[ciPosRefVelGiroMax + 1]);
                    //Las palabras de estado están formados por un solo byte
                    infoTurbina.bError1 = arrayTrama[ciPosError1];
                    infoTurbina.bError2 = arrayTrama[ciPosError2];
                    infoTurbina.bError3 = arrayTrama[ciPosError3];
                    infoTurbina.bEstado = arrayTrama[ciPosEstado];
                }
                else
                {
                    infoTurbina = null;
                }
            }
            catch
            {
                infoTurbina = null;
            }

            return infoTurbina;
        }
        #endregion
    }
}
```

2.1.8 TClienteSQL.cs

```
/************************
 ** TClienteSQL.cs **
    31/05/2016
 ***********************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;

namespace ComunicacionTurbina
{
    public class TClienteSQL
    {
        //Parámetros para la interacción con la base de datos
        private SqlConnection conexionSQL;
        private SqlCommand sqlComando;
        private string sFecha;

        public delegate void textToLogEventHandler(string text);
        public event textToLogEventHandler eventoTextToLog;

        #region Métodos
        public TClienteSQL()
        {
            conexionSQL = new
SqlConnection(Properties.Settings.Default.BBDD_Local_ConnString);
            sqlComando = new SqlCommand();
        }

        private void ConectarSQLServer()
        {//Conectar con la base de datos local
            try
            {
                conexionSQL.Open();
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("Conexión establecida con la base de datos.");
                }
                sqlComando.Connection = conexionSQL;
            }
            catch(Exception ex)
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("No se puede establecer conexión con la base de
datos: "+ ex.Message);
                }
            }
        }
```

```
        public void InsertaElementos(TInfoTurbina trama)
        {//Inserta en la tabla local los elementos de la trama recibida desde la turbina
            sFecha = DateTime.Now.ToString("dd-MM-yyyy HH:mm:ss.fff");

            sqlComando.CommandText =
                String.Format(
                "INSERT {0} (dtFecha, velGenerador, presionCamara, posicionCilindro,
potencia, vibraciones, corrienteIU, "
                + "corrienteIV, tensionVDC, temperatura1, temperatura2, comandoPosValvula,
refVelocidadGiroMax, "
                + "bError1, bError2, bError3, bEstado)"
                + "VALUES ('{1}', {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}, {11}, {12},
{13}, {14}, {15}, {16}, {17} )"
                , Properties.Settings.Default.sNombreTablaTurbina, sFecha,
trama.velGenerador, trama.presionCamara, trama.posicionCilindro, trama.potencia,
trama.vibraciones, trama.corrienteIU
                , trama.corrienteIV, trama.tensionVDC, trama.temperatura1,
trama.temperatura2, trama.comandoPosValvula, trama.refVelocidadGiroMax
                , trama.bError1, trama.bError2, trama.bError3, trama.bEstado);

            if (conexionSQL.State != System.Data.ConnectionState.Open)
            {//Si no hay conexión con la base de datos, intenta conectar
                this.ConectarSQLServer();
            }
            if (conexionSQL.State == System.Data.ConnectionState.Open)
            {//Ejecuta el comando para introducir los datos en la BBDD
                sqlComando.ExecuteNonQuery();
            }
            else
            {
                if (this.eventoTextToLog != null)
                {
                    this.eventoTextToLog("Ha ocurrido un problema en la conexión con la base
de datos.");
                }
            }
        }
        #endregion
    }

       }
```

2.1.9 TClienteEstadosTCP.cs

```
/*****************************
 ** TClienteEstadosTCP.cs **
    31/05/2016
 *****************************/
using System;
using System.Threading;
using System.Net;
```

```
using System.Net.Sockets;
using System.Drawing;

namespace ComunicacionTurbina
{
    public class TClienteEstadosTCP
    {
        private int iPuertoTCP_Estados; //Puerto en el que escucha el servidor de estados

        //Palabras que se van a enviar
        private byte byError1;
        private byte byError2;
        private byte byError3;
        private byte byEstado;

        private Thread hiloClienteEstados; //El hilo se crea, se lanza y se termina con cada
envío

        #region Métodos
        public TClienteEstadosTCP()
        {
        }

        public TClienteEstadosTCP(byte error1, byte error2, byte error3, byte estado)
        {
            iPuertoTCP_Estados = Properties.Settings.Default.iPuertoTCP_Estados;
            byError1 = error1;
            byError2 = error2;
            byError3 = error3;
            byEstado = estado;

            hiloClienteEstados = new Thread(new ThreadStart(this.EnviaEstados));
            hiloClienteEstados.Start();
        }

        private void EnviaEstados()
        {
            try
            {
                IPAddress[] ipv4Direcciones =
Array.FindAll(Dns.GetHostEntry(Properties.Settings.Default.ipServidorEstados).AddressList,
                                                        d => d.AddressFamily ==
AddressFamily.InterNetwork);

                IPAddress ipAddress = ipv4Direcciones[0];

                IPEndPoint listener = new IPEndPoint(ipAddress, iPuertoTCP_Estados);
//Parámetros del servidor de estados

                Socket sender = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);//Socket de envío

                sender.Connect(listener); //pide conexión

                byte[] msg = this.ComponeTramaEnviar();
                int iBytesEnviados = sender.Send(msg);
```

```
                Thread.Sleep(1000); //retardo para el envío

                //Desconexión
                sender.Shutdown(SocketShutdown.Both);
                sender.Close();
            }
            catch (Exception ex)
            {

            }
        }

        private byte[] ComponeTramaEnviar()
        {
            byte[] byTramaTcpEnviar = new byte[8];

            byTramaTcpEnviar[0] = 0x10;
            byTramaTcpEnviar[1] = 0x02;
            byTramaTcpEnviar[2] = byError1;
            byTramaTcpEnviar[3] = byError2;
            byTramaTcpEnviar[4] = byError3;
            byTramaTcpEnviar[5] = byEstado;
            byTramaTcpEnviar[6] = 0x10;
            byTramaTcpEnviar[7] = 0x03;

            return byTramaTcpEnviar;
        }
        #endregion
    }
}
```

2.1.10 TCheckeadorErrores.cs

```
/*****************************
 ** TCheckeadorErrores.cs **
    31/05/2016
 *****************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ComunicacionTurbina
{
    class TCheckeadorErrores
    {
        private byte byError1Anterior;
        private byte byError2Anterior;
        private byte byError3Anterior;
        private byte byEstadoAnterior;
```

```
        private bool bPrimerosDatos;

        public delegate void cambioEstadoErroresEventHandler(byte byErr1, byte byErr2, byte
byErr3, byte byEst);
        public event cambioEstadoErroresEventHandler eventoCambioEstadoErrores;

        #region Métodos
        public TCheckeadorErrores()
        {
            bPrimerosDatos = true;
        }

        public void CompruebaErroresYEstado(TInfoTurbina infoTurbina)
        {
            if (bPrimerosDatos == true)
            {//Captura de los valores de los bytes de estado y error en la primera captura
                byError1Anterior = infoTurbina.bError1;
                byError2Anterior = infoTurbina.bError2;
                byError3Anterior = infoTurbina.bError3;
                byEstadoAnterior = infoTurbina.bEstado;
                bPrimerosDatos = false;
            }

            if ((infoTurbina.bError1 != byError1Anterior) || (infoTurbina.bError2 !=
byError2Anterior) ||
                (infoTurbina.bError3 != byError3Anterior) || (infoTurbina.bEstado !=
byEstadoAnterior))
            {//Los bytes de estado de la captura anterior se comparan con los de la actual
para ver si se ha producido algún cambio
                //si se detecta cambio se envían los estados al HMI para que los interprete
                if (this.eventoCambioEstadoErrores != null)
                {
                    this.eventoCambioEstadoErrores(infoTurbina.bError1, infoTurbina.bError2,
infoTurbina.bError3, infoTurbina.bEstado);
                }

                byError1Anterior = infoTurbina.bError1;
                byError2Anterior = infoTurbina.bError2;
                byError3Anterior = infoTurbina.bError3;
                byEstadoAnterior = infoTurbina.bEstado;
            }
        }
        #endregion
    }
}
```

2.2 Código aplicación HMI

2.2.1 Program.cs

```
/*******************
 ** Program.cs **
    31/05/2016
 *******************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ClienteGraficas
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormMenu());
        }
    }
}
```

2.2.2 TServicioUI.cs

```
/************************
 ** TServicioUI.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Threading;
using System.Data.SqlClient;
using System.Windows.Forms;
using System.Data;
```

```
using Microsoft.Win32;
using DevExpress.XtraCharts;
using System.Drawing;

namespace ClienteGraficas
{
    class TServicioUI
    {
        public static TServicioUI instancia = null;

        private System.Windows.Forms.Timer tempo; //Temporización para la sincronización de
bases de datos
        private Thread hiloSincronizacion; //Hilo para la sincronización de bases de datos

        //Comandos y cadenas de conexión para la interacción con las bases de datos
        private SqlCommand sqlCommSelectFromTabla;
        private SqlCommand sqlCommActualizaFechaMax;
        private SqlConnection sqlConnRemota;
        private SqlConnection sqlConnLocal;

        private DateTime dtMaxFecha;
        private DataTable dTblNuevosDatos;

        public TServidorEstadosTCP servidorEstadosTCP;
        private TClienteComandosTCP clienteComandosTCP;

        public TConfigCurvas configCurvas = null; //Configuración actual de las curvas

        public delegate void delEstadoRecibidoHandler_IU(byte error1, byte error2, byte
error3, byte estado);

        public delegate void delNuevoEstado(byte byEstado);
        public delNuevoEstado eventoNuevoEstado;
        public delegate void delNuevosErrores(byte byError1, byte byError2, byte byError3);
        public event delNuevosErrores eventoNuevosErrores;

        #region Métodos
        public TServicioUI()
        {
            sqlCommSelectFromTabla = new SqlCommand();
            sqlCommActualizaFechaMax = new SqlCommand();
            sqlConnRemota = new
SqlConnection(Properties.Settings.Default.BBDD_Remota_ConnString);
            sqlConnLocal = new SqlConnection(Properties.Settings.Default.cadenaLocal);

            configCurvas = new TConfigCurvas();

            tempo = new System.Windows.Forms.Timer();
            tempo.Tick += new EventHandler(this.SincronizarTablas);

            //Sincronización inicial
            hiloSincronizacion = new Thread(new ThreadStart(HiloSincronizar));
            hiloSincronizacion.Start();
            tempo.Interval = /*(60 - DateTime.Now.Minute) * 60 * 1000 +*/ (60 -
DateTime.Now.Second) * 1000 + (1000 - DateTime.Now.Millisecond); //Siguiente actualización
después de una hora
```

```
            tempo.Start();

            dtMaxFecha = DateTime.Parse("1970-01-01T00:00:00.000");
            dTblNuevosDatos = new DataTable();

            servidorEstadosTCP = new TServidorEstadosTCP();

            servidorEstadosTCP.eventoMensajeRecibido += this.NuevoEstadoErroresRecibidos;
            servidorEstadosTCP.IniciarHiloServidor();

            this.AbreRegistro();//Abre el registro de la aplicación para obtener los valores
de configuración de las curvas
        }

        public static TServicioUI GetInstancia()
        {
            try
            {
                if (instancia == null)
                {
                    instancia = new TServicioUI();
                }
            }
            catch(Exception ex)
            {

            }

            return instancia;
        }

        private void NuevoEstadoErroresRecibidos(byte error1, byte error2, byte error3, byte
estado)
        {//Cuando se recibe un nuevo estado se interpreta cada una de sus palabras y se
muestran los formularios de alarmas y estado de la turbina
            if (this.eventoNuevoEstado != null)
            {
                this.eventoNuevoEstado(estado);
            }
            if (this.eventoNuevosErrores != null)
            {
                this.eventoNuevosErrores(error1, error2, error3);
            }
        }

        #region Sincronización BBDD
        private void HiloSincronizar()
        {//Sincronización de bases de datos. Cuando se lanza tempo.Tick += new
EventHandler(this.SincronizarTablas)
            try
            {//Consulta de la fecha del dato más reciente de la base de datos local
                sqlConnLocal.Open();
                sqlCommSelectFromTabla.Connection = sqlConnRemota;
            }
            catch (Exception ex)
            {
```

```
                MessageBox.Show("Error al establecer las conexiones con la base de datos
local: " + ex.Message, "Excepción", MessageBoxButtons.OK);
            }
            try
            {//Consulta de los datos recientes a la base de datos remota
                sqlConnRemota.Open();
                sqlCommActualizaFechaMax.Connection = sqlConnLocal;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al establecer las conexiones con la base de datos
remota: " + ex.Message, "Excepción", MessageBoxButtons.OK);
            }
            finally
            {//Actualización de la base de datos local con dichos valores
                this.SincronizaTablaLocal();

                sqlConnLocal.Close();
                sqlConnRemota.Close();
            }

        }

        private void SincronizaTablaLocal()
        {//Comprueba si la tabla local está vacía, si no lo está no hace nada
            try
            {
                SqlCommand cmdCompruebaTablaLocal = new SqlCommand("SELECT COUNT(*) FROM " +
Properties.Settings.Default.sNombreTablaLocal, sqlConnLocal);
                int count = Convert.ToInt32(cmdCompruebaTablaLocal.ExecuteScalar());

                if (count == 0)
                {//si la tabla local está vacía la rellena con todos los datos de la tabla
remota, inicialmente todas las fechas serán mayores que dtMaxFecha
                    this.SelectRowsFromTablaRemota();

                    if (dTblNuevosDatos.Rows.Count > 0)
                    {//Inserta los datos en la tabla local
                        this.InsertInTablaLocal();
                    }
                }
                else
                {//si la tabla local no está vacía
                    this.SelectMaxFechaTablaLocal();

                    this.SelectRowsFromTablaRemota();

                    if (dTblNuevosDatos.Rows.Count > 0)
                    {
                        this.InsertInTablaLocal();
                    }
                }
            }
            catch { }
        }
```

```
        private void SincronizarTablas(object sender, EventArgs e)
        {//Cada minuto se sincronizan las tablas remota y local
            //Establece el intervalo de actualización para el siguiente minuto
            tempo.Interval = /*(60 - DateTime.Now.Minute) * 60 * 1000 +*/ (60 -
DateTime.Now.Second) * 1000 + (1000 - DateTime.Now.Millisecond); //Siguiente actualización
después de una hora

            hiloSincronizacion = new Thread(new ThreadStart(HiloSincronizar));
            hiloSincronizacion.Start();
        }

        private void SelectMaxFechaTablaLocal()
        {//Selecciona la máxima fecha de la tabla local
            try
            {
                sqlCommActualizaFechaMax.CommandText = "SELECT MAX(dtFecha) FROM " +
Properties.Settings.Default.sNombreTablaLocal;
                dtMaxFecha = (DateTime)sqlCommActualizaFechaMax.ExecuteScalar();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al consultar la BBDD local. " + ex.Message);
            }
        }

        private void SelectRowsFromTablaRemota()
        {//Selecciona las filas de la tabla remota cuya fecha es mayor que la máxima de la
tabla local y las mete en una data table
            try
            {
                sqlCommSelectFromTabla.CommandText =
                    "SELECT * FROM " + Properties.Settings.Default.sNombreTablaRemota + "
WHERE dtFecha>'" + dtMaxFecha.ToString("yyyy-MM-ddTHH:mm:ss.fff") + "'";

                SqlDataAdapter sqlAdapterConsulta = new
SqlDataAdapter(sqlCommSelectFromTabla);
                dTblNuevosDatos.Clear();
                sqlAdapterConsulta.Fill(dTblNuevosDatos);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al consultar la BBDD remota. " + ex.Message);
            }
        }

        private void InsertInTablaLocal()
        {//Inserta los datos del dataTable en tabla local
            try
            {
                using (var bulkCopy = new
SqlBulkCopy(Properties.Settings.Default.cadenaLocal, SqlBulkCopyOptions.KeepIdentity))
                {
                    foreach (DataColumn col in dTblNuevosDatos.Columns)
                    {
                        bulkCopy.ColumnMappings.Add(col.ColumnName, col.ColumnName);
                    }
```

```
                    bulkCopy.BulkCopyTimeout = 600;
                    bulkCopy.DestinationTableName =
Properties.Settings.Default.sNombreTablaLocal;
                    bulkCopy.WriteToServer(dTblNuevosDatos);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al consultar la BBDD local. " + ex.Message);
            }
        }
        #endregion

        #region Manejo del registro de Windows
        private void AbreRegistro()
        {//Abre o crea los registros correspondientes para adquirir o crear los datos de los
ejes y las series
            RegistryKey regKeySoftware = Registry.CurrentUser.OpenSubKey("Software", true);
            RegistryKey regKeyITRESA = regKeySoftware.OpenSubKey("ITRESA", true);

            this.AbreRegistroITRESA(regKeySoftware, regKeyITRESA);

            if (regKeySoftware != null)
            {
                regKeySoftware.Close();
            }
            if (regKeyITRESA != null)
            {
                regKeyITRESA.Close();
            }
        }

        private void AbreRegistroITRESA(RegistryKey regSoft, RegistryKey regITRESA)
        {//Abre o crea los registros ITRESA y de la aplicación, con sus correspondientes
registros para cada eje y para cada serie
            RegistryKey regApp = null;

            if (regSoft != null)
            {
                if (regITRESA == null)
                {
                    regITRESA = regSoft.CreateSubKey("ITRESA",
RegistryKeyPermissionCheck.ReadWriteSubTree);
                }
                else
                {
                    regApp = regITRESA.OpenSubKey("App_SCADA_TURBINA", true);
                }
                if (regApp == null)
                {
                    regApp = regITRESA.CreateSubKey("App_SCADA_TURBINA",
RegistryKeyPermissionCheck.ReadWriteSubTree);
                }
                //Registros ejes
                this.AbreRegistroEje("EjeVelocidades", regApp, ref configCurvas.confEjeVel);
                this.AbreRegistroEje("EjePresion", regApp, ref configCurvas.confEjePres);
```

```
                this.AbreRegistroEje("EjePosicionCilindro", regApp, ref
configCurvas.confEjePosCil);
                this.AbreRegistroEje("EjePotencia", regApp, ref configCurvas.confEjePot);
                this.AbreRegistroEje("EjeVibraciones", regApp, ref
configCurvas.confEjeVibr);
                this.AbreRegistroEje("EjeCorrientes", regApp, ref configCurvas.confEjeCorr);
                this.AbreRegistroEje("EjeTensionVDC", regApp, ref configCurvas.confEjeTens);
                this.AbreRegistroEje("EjeTemperaturas", regApp, ref
configCurvas.confEjeTemp);
                this.AbreRegistroEje("EjeComandoPosValvula", regApp, ref
configCurvas.confEjeComPos);
                //Registros series
                this.AbreRegistroSerie("SerieVelocidadGenerador", regApp,
configCurvas.serieVelGenerador);
                this.AbreRegistroSerie("SeriePresionCamara", regApp,
configCurvas.seriePresCamara);
                this.AbreRegistroSerie("SeriePosicionCilindro", regApp,
configCurvas.seriePosCilindro);
                this.AbreRegistroSerie("SeriePotencia", regApp, configCurvas.seriePotencia);
                this.AbreRegistroSerie("SerieVibraciones", regApp,
configCurvas.serieVibraciones);
                this.AbreRegistroSerie("SerieCorrienteIU", regApp,
configCurvas.serieCorrienteIU);
                this.AbreRegistroSerie("SerieCorrienteIV", regApp,
configCurvas.serieCorrienteIV);
                this.AbreRegistroSerie("SerieTensionVDC", regApp,
configCurvas.serieTensionVDC);
                this.AbreRegistroSerie("SerieTemp1", regApp, configCurvas.serieTemp1);
                this.AbreRegistroSerie("SerieTemp2", regApp, configCurvas.serieTemp2);
                this.AbreRegistroSerie("SerieComandoPosValvula", regApp,
configCurvas.serieComPosValvula);
                this.AbreRegistroSerie("SerieRefVelGiroMax", regApp,
configCurvas.serieRefVelMax);
            }
        }

        private void AbreRegistroEje(string sNombreEje, RegistryKey regApp, ref TConfEjeY
confEje)
        {//Abre o crea el registro para el eje correspondiente
            RegistryKey regEje = null;
            try
            {
                regEje = regApp.OpenSubKey(sNombreEje, true);
                if (regEje == null)
                {
                    regEje = regApp.CreateSubKey(sNombreEje,
RegistryKeyPermissionCheck.ReadWriteSubTree);
                    if (regEje != null)
                    {
                        regEje.SetValue("Minimo", confEje.minimoY);
                        regEje.SetValue("Maximo", confEje.maximoY);
                    }
                }
                else
                {
                    confEje.minimoY = Convert.ToSingle(regEje.GetValue("Minimo"));
```

```
                    confEje.maximoY = Convert.ToSingle(regEje.GetValue("Maximo"));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al abrir el registro " + sNombreEje + ". " +
ex.Message);
            }
        }

        private void AbreRegistroSerie(string sNombreSerie, RegistryKey regApp, Series
serieRegistro)
        {//Abre o crea el registro para la serie correspondiente
            RegistryKey regSerie = null;
            try
            {
                regSerie = regApp.OpenSubKey(sNombreSerie, true);
                if (regSerie == null)
                {
                    regSerie = regApp.CreateSubKey(sNombreSerie,
RegistryKeyPermissionCheck.ReadWriteSubTree);

                    if (regSerie != null)
                    {
                        regSerie.SetValue("Visible", serieRegistro.Visible);
                        regSerie.SetValue("Color", serieRegistro.View.Color.ToArgb());
                    }
                }
                else
                {
                    serieRegistro.Visible = Convert.ToBoolean(regSerie.GetValue("Visible"));
                    serieRegistro.View.Color =
Color.FromArgb(Convert.ToInt32(regSerie.GetValue("Color")));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al abrir el registro " + sNombreSerie + ". " +
ex.Message);
            }
        }

        public void EnvioDeComando(byte com, short refe)
        {
            clienteComandosTCP = new TClienteComandosTCP(com, refe);
        }
        #endregion
        #endregion
    }
}
```

2.2.3 FormMenu.cs

```
/************************
 ** TServicioUI.cs **
    31/05/2016
 ***********************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Threading;
using System.Data.SqlClient;
using System.Windows.Forms;
using System.Data;
using Microsoft.Win32;
using DevExpress.XtraCharts;
using System.Drawing;

namespace ClienteGraficas
{
    class TServicioUI
    {
        public static TServicioUI instancia = null;

        private System.Windows.Forms.Timer tempo; //Temporización para la sincronización de
bases de datos
        private Thread hiloSincronizacion; //Hilo para la sincronización de bases de datos

        //Comandos y cadenas de conexión para la interacción con las bases de datos
        private SqlCommand sqlCommSelectFromTabla;
        private SqlCommand sqlCommActualizaFechaMax;
        private SqlConnection sqlConnRemota;
        private SqlConnection sqlConnLocal;

        private DateTime dtMaxFecha;
        private DataTable dTblNuevosDatos;

        public TServidorEstadosTCP servidorEstadosTCP;
        private TClienteComandosTCP clienteComandosTCP;

        public TConfigCurvas configCurvas = null; //Configuración actual de las curvas

        public delegate void delEstadoRecibidoHandler_IU(byte error1, byte error2, byte
error3, byte estado);

        public delegate void delNuevoEstado(byte byEstado);
        public delNuevoEstado eventoNuevoEstado;
        public delegate void delNuevosErrores(byte byError1, byte byError2, byte byError3);
        public event delNuevosErrores eventoNuevosErrores;

        #region Métodos
        public TServicioUI()
        {
```

```
            sqlCommSelectFromTabla = new SqlCommand();
            sqlCommActualizaFechaMax = new SqlCommand();
            sqlConnRemota = new
SqlConnection(Properties.Settings.Default.BBDD_Remota_ConnString);
            sqlConnLocal = new SqlConnection(Properties.Settings.Default.cadenaLocal);

            configCurvas = new TConfigCurvas();

            tempo = new System.Windows.Forms.Timer();
            tempo.Tick += new EventHandler(this.SincronizarTablas);

            //Sincronización inicial
            hiloSincronizacion = new Thread(new ThreadStart(HiloSincronizar));
            hiloSincronizacion.Start();
            tempo.Interval = /*(60 - DateTime.Now.Minute) * 60 * 1000 +*/ (60 -
DateTime.Now.Second) * 1000 + (1000 - DateTime.Now.Millisecond); //Siguiente actualización
después de una hora
            tempo.Start();

            dtMaxFecha = DateTime.Parse("1970-01-01T00:00:00.000");
            dTblNuevosDatos = new DataTable();

            servidorEstadosTCP = new TServidorEstadosTCP();

            servidorEstadosTCP.eventoMensajeRecibido += this.NuevoEstadoErroresRecibidos;
            servidorEstadosTCP.IniciarHiloServidor();

            this.AbreRegistro();//Abre el registro de la aplicación para obtener los valores
de configuración de las curvas
        }

        public static TServicioUI GetInstancia()
        {
            try
            {
                if (instancia == null)
                {
                    instancia = new TServicioUI();
                }
            }
            catch(Exception ex)
            {

            }

            return instancia;
        }

        private void NuevoEstadoErroresRecibidos(byte error1, byte error2, byte error3, byte
estado)
        {//Cuando se recibe un nuevo estado se interpreta cada una de sus palabras y se
muestran los formularios de alarmas y estado de la turbina
            if (this.eventoNuevoEstado != null)
            {
                this.eventoNuevoEstado(estado);
            }
```

```
            if (this.eventoNuevosErrores != null)
            {
                this.eventoNuevosErrores(error1, error2, error3);
            }
        }

        #region Sincronización BBDD
        private void HiloSincronizar()
        {//Sincronización de bases de datos. Cuando se lanza tempo.Tick += new
EventHandler(this.SincronizarTablas)
            try
            {//Consulta de la fecha del dato más reciente de la base de datos local
                sqlConnLocal.Open();
                sqlCommSelectFromTabla.Connection = sqlConnRemota;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al establecer las conexiones con la base de datos
local: " + ex.Message, "Excepción", MessageBoxButtons.OK);
            }
            try
            {//Consulta de los datos recientes a la base de datos remota
                sqlConnRemota.Open();
                sqlCommActualizaFechaMax.Connection = sqlConnLocal;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al establecer las conexiones con la base de datos
remota: " + ex.Message, "Excepción", MessageBoxButtons.OK);
            }
            finally
            {//Actualización de la base de datos local con dichos valores
                this.SincronizaTablaLocal();

                sqlConnLocal.Close();
                sqlConnRemota.Close();
            }

        }

        private void SincronizaTablaLocal()
        {//Comprueba si la tabla local está vacía, si no lo está no hace nada
            try
            {
                SqlCommand cmdCompruebaTablaLocal = new SqlCommand("SELECT COUNT(*) FROM " +
Properties.Settings.Default.sNombreTablaLocal, sqlConnLocal);
                int count = Convert.ToInt32(cmdCompruebaTablaLocal.ExecuteScalar());

                if (count == 0)
                {//si la tabla local está vacía la rellena con todos los datos de la tabla
remota, inicialmente todas las fechas serán mayores que dtMaxFecha
                    this.SelectRowsFromTablaRemota();

                    if (dTblNuevosDatos.Rows.Count > 0)
                    {//Inserta los datos en la tabla local
                        this.InsertInTablaLocal();
```

```
                }
            }
            else
            {//si la tabla local no está vacía
                this.SelectMaxFechaTablaLocal();

                this.SelectRowsFromTablaRemota();

                if (dTblNuevosDatos.Rows.Count > 0)
                {
                    this.InsertInTablaLocal();
                }
            }
        }
        catch { }
    }

    private void SincronizarTablas(object sender, EventArgs e)
    {//Cada minuto se sincronizan las tablas remota y local
        //Establece el intervalo de actualización para el siguiente minuto
        tempo.Interval = /*(60 - DateTime.Now.Minute) * 60 * 1000 +*/ (60 -
DateTime.Now.Second) * 1000 + (1000 - DateTime.Now.Millisecond); //Siguiente actualización
después de una hora

        hiloSincronizacion = new Thread(new ThreadStart(HiloSincronizar));
        hiloSincronizacion.Start();
    }

    private void SelectMaxFechaTablaLocal()
    {//Selecciona la máxima fecha de la tabla local
        try
        {
            sqlCommActualizaFechaMax.CommandText = "SELECT MAX(dtFecha) FROM " +
Properties.Settings.Default.sNombreTablaLocal;
            dtMaxFecha = (DateTime)sqlCommActualizaFechaMax.ExecuteScalar();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al consultar la BBDD local. " + ex.Message);
        }
    }

    private void SelectRowsFromTablaRemota()
    {//Selecciona las filas de la tabla remota cuya fecha es mayor que la máxima de la
tabla local y las mete en una data table
        try
        {
            sqlCommSelectFromTabla.CommandText =
                "SELECT * FROM " + Properties.Settings.Default.sNombreTablaRemota + "
WHERE dtFecha>'" + dtMaxFecha.ToString("yyyy-MM-ddTHH:mm:ss.fff") + "'";

            SqlDataAdapter sqlAdapterConsulta = new
SqlDataAdapter(sqlCommSelectFromTabla);
            dTblNuevosDatos.Clear();
            sqlAdapterConsulta.Fill(dTblNuevosDatos);
        }
```

```
            catch (Exception ex)
            {
                MessageBox.Show("Error al consultar la BBDD remota. " + ex.Message);
            }
        }

        private void InsertInTablaLocal()
        {//Inserta los datos del dataTable en tabla local
            try
            {
                using (var bulkCopy = new
SqlBulkCopy(Properties.Settings.Default.cadenaLocal, SqlBulkCopyOptions.KeepIdentity))
                {
                    foreach (DataColumn col in dTblNuevosDatos.Columns)
                    {
                        bulkCopy.ColumnMappings.Add(col.ColumnName, col.ColumnName);
                    }
                    bulkCopy.BulkCopyTimeout = 600;
                    bulkCopy.DestinationTableName =
Properties.Settings.Default.sNombreTablaLocal;
                    bulkCopy.WriteToServer(dTblNuevosDatos);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al consultar la BBDD local. " + ex.Message);
            }
        }
        #endregion

        #region Manejo del registro de Windows
        private void AbreRegistro()
        {//Abre o crea los registros correspondientes para adquirir o crear los datos de los
ejes y las series
            RegistryKey regKeySoftware = Registry.CurrentUser.OpenSubKey("Software", true);
            RegistryKey regKeyITRESA = regKeySoftware.OpenSubKey("ITRESA", true);

            this.AbreRegistroITRESA(regKeySoftware, regKeyITRESA);

            if (regKeySoftware != null)
            {
                regKeySoftware.Close();
            }
            if (regKeyITRESA != null)
            {
                regKeyITRESA.Close();
            }
        }

        private void AbreRegistroITRESA(RegistryKey regSoft, RegistryKey regITRESA)
        {//Abre o crea los registros ITRESA y de la aplicación, con sus correspondientes
registros para cada eje y para cada serie
            RegistryKey regApp = null;

            if (regSoft != null)
            {
```

```
                if (regITRESA == null)
                {
                    regITRESA = regSoft.CreateSubKey("ITRESA",
RegistryKeyPermissionCheck.ReadWriteSubTree);
                }
                else
                {
                    regApp = regITRESA.OpenSubKey("App_SCADA_TURBINA", true);
                }
                if (regApp == null)
                {
                    regApp = regITRESA.CreateSubKey("App_SCADA_TURBINA",
RegistryKeyPermissionCheck.ReadWriteSubTree);
                }
                //Registros ejes
                this.AbreRegistroEje("EjeVelocidades", regApp, ref configCurvas.confEjeVel);
                this.AbreRegistroEje("EjePresion", regApp, ref configCurvas.confEjePres);
                this.AbreRegistroEje("EjePosicionCilindro", regApp, ref
configCurvas.confEjePosCil);
                this.AbreRegistroEje("EjePotencia", regApp, ref configCurvas.confEjePot);
                this.AbreRegistroEje("EjeVibraciones", regApp, ref
configCurvas.confEjeVibr);
                this.AbreRegistroEje("EjeCorrientes", regApp, ref configCurvas.confEjeCorr);
                this.AbreRegistroEje("EjeTensionVDC", regApp, ref configCurvas.confEjeTens);
                this.AbreRegistroEje("EjeTemperaturas", regApp, ref
configCurvas.confEjeTemp);
                this.AbreRegistroEje("EjeComandoPosValvula", regApp, ref
configCurvas.confEjeComPos);
                //Registros series
                this.AbreRegistroSerie("SerieVelocidadGenerador", regApp,
configCurvas.serieVelGenerador);
                this.AbreRegistroSerie("SeriePresionCamara", regApp,
configCurvas.seriePresCamara);
                this.AbreRegistroSerie("SeriePosicionCilindro", regApp,
configCurvas.seriePosCilindro);
                this.AbreRegistroSerie("SeriePotencia", regApp, configCurvas.seriePotencia);
                this.AbreRegistroSerie("SerieVibraciones", regApp,
configCurvas.serieVibraciones);
                this.AbreRegistroSerie("SerieCorrienteIU", regApp,
configCurvas.serieCorrienteIU);
                this.AbreRegistroSerie("SerieCorrienteIV", regApp,
configCurvas.serieCorrienteIV);
                this.AbreRegistroSerie("SerieTensionVDC", regApp,
configCurvas.serieTensionVDC);
                this.AbreRegistroSerie("SerieTemp1", regApp, configCurvas.serieTemp1);
                this.AbreRegistroSerie("SerieTemp2", regApp, configCurvas.serieTemp2);
                this.AbreRegistroSerie("SerieComandoPosValvula", regApp,
configCurvas.serieComPosValvula);
                this.AbreRegistroSerie("SerieRefVelGiroMax", regApp,
configCurvas.serieRefVelMax);
            }
        }

        private void AbreRegistroEje(string sNombreEje, RegistryKey regApp, ref TConfEjeY
confEje)
        {//Abre o crea el registro para el eje correspondiente
```

```
            RegistryKey regEje = null;
            try
            {
                regEje = regApp.OpenSubKey(sNombreEje, true);
                if (regEje == null)
                {
                    regEje = regApp.CreateSubKey(sNombreEje,
RegistryKeyPermissionCheck.ReadWriteSubTree);
                    if (regEje != null)
                    {
                        regEje.SetValue("Minimo", confEje.minimoY);
                        regEje.SetValue("Maximo", confEje.maximoY);
                    }
                }
                else
                {
                    confEje.minimoY = Convert.ToSingle(regEje.GetValue("Minimo"));
                    confEje.maximoY = Convert.ToSingle(regEje.GetValue("Maximo"));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al abrir el registro " + sNombreEje + ". " +
ex.Message);
            }
        }

        private void AbreRegistroSerie(string sNombreSerie, RegistryKey regApp, Series
serieRegistro)
        {//Abre o crea el registro para la serie correspondiente
            RegistryKey regSerie = null;
            try
            {
                regSerie = regApp.OpenSubKey(sNombreSerie, true);
                if (regSerie == null)
                {
                    regSerie = regApp.CreateSubKey(sNombreSerie,
RegistryKeyPermissionCheck.ReadWriteSubTree);

                    if (regSerie != null)
                    {
                        regSerie.SetValue("Visible", serieRegistro.Visible);
                        regSerie.SetValue("Color", serieRegistro.View.Color.ToArgb());
                    }
                }
                else
                {
                    serieRegistro.Visible = Convert.ToBoolean(regSerie.GetValue("Visible"));
                    serieRegistro.View.Color =
Color.FromArgb(Convert.ToInt32(regSerie.GetValue("Color")));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al abrir el registro " + sNombreSerie + ". " +
ex.Message);
```

```
            }
        }

        public void EnvioDeComando(byte com, short refe)
        {
            clienteComandosTCP = new TClienteComandosTCP(com, refe);
        }
        #endregion
        #endregion
    }
}
```

2.2.4 FormAlarmas.cs

```
/************************
 ** FormAlarmas.cs **
      31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace ClienteGraficas
{
    public partial class FormAlarmas : Form
    {
        public delegate void delSetText(string sText, Color col); //Delegado para mostrar
texto en el log
        delegate void delInterpretarErrores(byte byError1, byte byError2, byte byError3);

        private StreamWriter streamWriter;
        private TServicioUI servicio;
        #region Métodos
        public FormAlarmas()
        {
            InitializeComponent();
            servicio = TServicioUI.GetInstancia();

            servicio.eventoNuevosErrores += this.InterpretarErrores;
        }

        #region Eventos del formulario
        private void FormAlarmas_Load(object sender, EventArgs e)
        {
            this.Icon = Properties.Resources.icono;
```

```
        }

        private void FormAlarmas_FormClosing(object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }
        #endregion

        #region Interpretación de errores
        public void Invoke_InterpretarErrores(byte byError1, byte byError2, byte byError3)
        {//De servicio.eventoNuevosErrores += this.InterpretarErrores;
            this.InterpretaError1(byError1);
            this.InterpretaError2(byError2);
            this.InterpretaError3(byError3);
            //se interpreta el estado de las palabras de error y se muestra la ventana de
alarmas
            this.Show();
            this.BringToFront();
        }

        private void InterpretarErrores(byte byError1, byte byError2, byte byError3)
        {
            if (this.InvokeRequired == true)
            {
                Invoke(new delInterpretarErrores(this.Invoke_InterpretarErrores), new
object[] { byError1, byError2, byError3 });
            }
            else
            {
                Invoke_InterpretarErrores(byError1, byError2, byError3);
            }
        }

        private void InterpretaError1(byte byError1)
        {//Interpreta la palabra de error1 y, en función de sus bits, modifica los labels
correspondientes
            System.Collections.BitArray bitArrError1 = new System.Collections.BitArray(new
byte[] { byError1 });

            if (bitArrError1[0] == true)
            {
                this.lblErrSensorIU.Text = "Lectura anómala";
                this.lblSensorIU.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor IU: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorIU.Text = "Correcto";
                this.lblSensorIU.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor IU: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[1] == true)
            {
                this.lblErrSensorIV.Text = "Lectura anómala";
```

```
                this.lblSensorIV.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor IV: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorIV.Text = "Correcto";
                this.lblSensorIV.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor IV: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[2] == true)
            {
                this.lblErrSensorIR.Text = "Lectura anómala";
                this.lblSensorIR.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor IR: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorIR.Text = "Correcto";
                this.lblSensorIR.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor IR: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[3] == true)
            {
                this.lblErrSensorIS.Text = "Lectura anómala";
                this.lblSensorIS.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor IS: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorIS.Text = "Correcto";
                this.lblSensorIS.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor IS: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[4] == true)
            {
                this.lblErrSensorVDC.Text = "Lectura anómala";
                this.lblSensorVDC.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor VDC: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorVDC.Text = "Correcto";
                this.lblSensorVDC.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor VDC: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[5] == true)
            {
                this.lblErrSensorVibraciones.Text = "Lectura anómala";
                this.lblSensorVibraciones.Appearance.Image =
Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor vibraciones: Lectura anómala", Color.Red);
            }
```

```
            else
            {
                this.lblErrSensorVibraciones.Text = "Correcto";
                this.lblSensorVibraciones.Appearance.Image =
Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor vibraciones: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[6] == true)
            {
                this.lblErrSensorPosCilindro.Text = "Lectura anómala";
                this.lblSensorPosCilindro.Appearance.Image =
Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor posición cilindro: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorPosCilindro.Text = "Correcto";
                this.lblSensorPosCilindro.Appearance.Image =
Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor posición cilindro: Correcto", Color.YellowGreen);
            }

            if (bitArrError1[7] == true)
            {
                this.lblErrSensorPresion.Text = "Lectura anómala";
                this.lblSensorPresion.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor presión: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorPresion.Text = "Correcto";
                this.lblSensorPresion.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor presión: Correcto", Color.YellowGreen);
            }
        }

        private void InterpretaError2(byte byError2)
        {//Interpreta la palabra de error2 y, en función de sus bits, modifica los labels
correspondientes
            System.Collections.BitArray bitArrError2 = new System.Collections.BitArray(new
byte[] { byError2 });

            if (bitArrError2[0] == true)
            {
                this.lblErrSensorTemp1.Text = "Lectura anómala";
                this.lblSensorTemp1.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Sensor temperatura 1: Lectura anómala", Color.Red);
            }
            else
            {
                this.lblErrSensorTemp1.Text = "Correcto";
                this.lblSensorTemp1.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Sensor temperatura 1: Correcto", Color.YellowGreen);
            }
```

```
if (bitArrError2[1] == true)
{
    this.lblErrSensorTemp2.Text = "Lectura anómala";
    this.lblSensorTemp2.Appearance.Image = Properties.Resources.cancel_16x16;
    this.AddTextoLog("Sensor temperatura 2: Lectura anómala", Color.Red);
}
else
{
    this.lblErrSensorTemp2.Text = "Correcto";
    this.lblSensorTemp2.Appearance.Image = Properties.Resources.apply_16x16;
    this.AddTextoLog("Sensor temperatura 2: Correcto", Color.YellowGreen);
}

if (bitArrError2[2] == true)
{
    this.lblErrFaseU.Text = "Sobrecorriente";
    this.lblFaseU.Appearance.Image = Properties.Resources.cancel_16x16;
    this.AddTextoLog("Fase U: Sobrecorriente", Color.Red);
}
else
{
    this.lblErrFaseU.Text = "Correcto";
    this.lblFaseU.Appearance.Image = Properties.Resources.apply_16x16;
    this.AddTextoLog("Fase U: Correcto", Color.YellowGreen);
}

if (bitArrError2[3] == true)
{
    this.lblErrFaseV.Text = "Sobrecorriente";
    this.lblFaseV.Appearance.Image = Properties.Resources.cancel_16x16;
    this.AddTextoLog("Fase V: Sobrecorriente", Color.Red);
}
else
{
    this.lblErrFaseV.Text = "Correcto";
    this.lblFaseV.Appearance.Image = Properties.Resources.apply_16x16;
    this.AddTextoLog("Fase V: Correcto", Color.YellowGreen);
}

if (bitArrError2[4] == true)
{
    this.lblErrFaseR.Text = "Sobrecorriente";
    this.lblFaseR.Appearance.Image = Properties.Resources.cancel_16x16;
    this.AddTextoLog("Fase R: Sobrecorriente", Color.Red);
}
else
{
    this.lblErrFaseR.Text = "Correcto";
    this.lblFaseR.Appearance.Image = Properties.Resources.apply_16x16;
    this.AddTextoLog("Fase R: Correcto", Color.YellowGreen);
}

if (bitArrError2[5] == true)
{
    this.lblErrFaseS.Text = "Sobrecorriente";
    this.lblFaseS.Appearance.Image = Properties.Resources.cancel_16x16;
```

```
                this.AddTextoLog("Fase S: Sobrecorriente", Color.Red);
            }
            else
            {
                this.lblErrFaseS.Text = "Correcto";
                this.lblFaseS.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Fase S: Correcto", Color.YellowGreen);
            }

            if (bitArrError2[6] == true)
            {
                this.lblErrVibracion.Text = "Vibración excesiva";
                this.lblVibracion.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Vibración: Vibración excesiva", Color.Red);
            }
            else
            {
                this.lblErrVibracion.Text = "Correcto";
                this.lblVibracion.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Vibración: Correcto", Color.YellowGreen);
            }

            if (bitArrError2[7] == true)
            {
                this.lblErrorPresion.Text = "Presión excesiva";
                this.lblPresion.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Presión: Presión excesiva", Color.Red);
            }
            else
            {
                this.lblErrorPresion.Text = "Correcto";
                this.lblPresion.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Presión: Correcto", Color.YellowGreen);
            }
        }

        private void InterpretaError3(byte byError3)
        {//Interpreta la palabra de error3 y, en función de sus bits, modifica los labels
correspondientes
            System.Collections.BitArray bitArrError3 = new System.Collections.BitArray(new
byte[] { byError3 });

            if (bitArrError3[0] == true)
            {
                this.lblErrorVelocidad.Text = "Velocidad excesiva";
                this.lblVelocidad.Appearance.Image = Properties.Resources.cancel_16x16;
                this.AddTextoLog("Velocidad: Velocidad excesiva", Color.Red);
            }
            else
            {
                this.lblErrorVelocidad.Text = "Correcto";
                this.lblVelocidad.Appearance.Image = Properties.Resources.apply_16x16;
                this.AddTextoLog("Velocidad: Correcto", Color.YellowGreen);
            }

            if (bitArrError3[1] == true)
```

```
        {
            this.lblErrorBusDC.Text = "Sobretensión bus DC";
            this.lblBusDC.Appearance.Image = Properties.Resources.cancel_16x16;
            this.AddTextoLog("Bus DC: Sobretensión", Color.Red);
        }
        else
        {
            this.lblErrorBusDC.Text = "Correcto";
            this.lblBusDC.Appearance.Image = Properties.Resources.apply_16x16;
            this.AddTextoLog("Bus DC: Correcto", Color.YellowGreen);
        }

        if (bitArrError3[2] == true)
        {
            this.lblErrorEncoder.Text = "En fallo";
            this.lblEncoder.Appearance.Image = Properties.Resources.cancel_16x16;
            this.AddTextoLog("Encoder: En fallo", Color.Red);
        }
        else
        {
            this.lblErrorEncoder.Text = "Correcto";
            this.lblEncoder.Appearance.Image = Properties.Resources.apply_16x16;
            this.AddTextoLog("Encoder: Correcto", Color.YellowGreen);
        }

        if (bitArrError3[3] == true)
        {
            this.lblErrorTemp1.Text = "Excesiva";
            this.lblTemp1.Appearance.Image = Properties.Resources.cancel_16x16;
            this.AddTextoLog("Temperatura 1: Excesiva", Color.Red);
        }
        else
        {
            this.lblErrorTemp1.Text = "Correcto";
            this.lblTemp1.Appearance.Image = Properties.Resources.apply_16x16;
            this.AddTextoLog("Temperatura 1: Correcto", Color.YellowGreen);
        }

        if (bitArrError3[4] == true)
        {
            this.lblErrorTemp2.Text = "Excesiva";
            this.lblTemp2.Appearance.Image = Properties.Resources.cancel_16x16;
            this.AddTextoLog("Temperatura 2: Excesiva", Color.Red);
        }
        else
        {
            this.lblErrorTemp2.Text = "Correcto";
            this.lblTemp2.Appearance.Image = Properties.Resources.apply_16x16;
            this.AddTextoLog("Temperatura 2: Correcto", Color.YellowGreen);
        }
    }
    #endregion

    private void AddTextoLog(string sTexto, Color color)
    {//Mostrar los eventos correspondientes en el log de alarmas
        if (lvLogAlarmas.Items.Count >= 200)
```

```
            {//El log muestra 200 elementos como máximo
                lvLogAlarmas.Items.RemoveAt(199);
            }

            ListViewItem lvItem = new ListViewItem();

            lvItem.Text = DateTime.Now.ToString("dd/mm/yyyy HH:mm:ss.fff");
            lvItem.SubItems.Add(sTexto);
            lvItem.SubItems[1].BackColor = color;
            lvItem.UseItemStyleForSubItems = false;
            lvLogAlarmas.Items.Insert(0, lvItem); //Inserta elementos en el log siempre en
la primera línea desplazando el resto hacia abajo
            this.AgregarLineaLog(lvItem.Text + "\t|||\t" + lvItem.SubItems[1].Text);
//Agrega una nueva línea al log.txt
        }

        private void AgregarLineaLog(string sNuevaLinea)
        {
            try
            {
                streamWriter = new StreamWriter(@Properties.Settings.Default.rutaLogAlarmas,
true);
                streamWriter.WriteLine(sNuevaLinea);
                streamWriter.Close();
            }
            catch (Exception ex)
            {
                //this.MuestraTexto(ex.Message);
            }
        }
        #endregion
    }
}
```

2.2.5 FormEstado.cs

```
/************************
 ** FormEstado.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace ClienteGraficas
```

```
{
    public partial class FormEstado : Form
    {
        public delegate void delSetText(string sText, Color col); //Delegado para mostrar
texto en el log
        delegate void delInterpretarEstado(byte byEstado);
        private TServicioUI servicio;
        private StreamWriter streamWriter;

        #region Métodos
        public FormEstado()
        {
            InitializeComponent();
            servicio = TServicioUI.GetInstancia();

            servicio.eventoNuevoEstado += this.InterpretaEstado;
        }

        #region Eventos del formulario
        private void FormEstado_Load(object sender, EventArgs e)
        {
            this.Icon = Properties.Resources.icono;
        }

        private void FormEstado_FormClosing(object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }
        #endregion

        #region Interpretación de estados
        private void Invoke_InterpretaEstado(byte byEstado)
        {//Interpreta la palabra de estado y actualiza los labels en función de cada bit
            System.Collections.BitArray bitArrEstado = new System.Collections.BitArray(new
byte[] { byEstado });

            if (bitArrEstado[0] == true)
            {
                this.lblVelocidadLimitada.Text = "Sí";
                this.AddTextoLog("Velocidad limitada: Sí", Color.White);
            }
            else
            {
                this.lblVelocidadLimitada.Text = "No";
                this.AddTextoLog("Velocidad limitada: No", Color.White);
            }

            if (bitArrEstado[1] == true)
            {
                this.lblPosicionValvula.Text = "Activado";
                this.AddTextoLog("Posición intermedia válvula: Activado", Color.White);
            }
            else
            {
                this.lblPosicionValvula.Text = "Desactivado";
```

```
        this.AddTextoLog("Posición intermedia válvula: Desactivado", Color.White);
    }

    if (bitArrEstado[2] == true)
    {
        this.lblEstadoFuncionamiento.Text = "Parado";
        this.AddTextoLog("Estado funcionamiento: Parado", Color.White);
    }
    else
    {
        this.lblEstadoFuncionamiento.Text = "Normal";
        this.AddTextoLog("Estado funcionamiento: Normal", Color.White);
    }

    if (bitArrEstado[3] == true)
    {
        this.lblEstadoAlarma.Text = "En alarma";
        this.lblEstadoAlarma.Appearance.Image = Properties.Resources.info_16x16;
        this.AddTextoLog("Estado alarma: En alarma", Color.Red);
    }
    else
    {
        this.lblEstadoAlarma.Text = "Funcionamiento OK";
        this.lblEstadoAlarma.Appearance.Image = null;
        this.AddTextoLog("Estado alarma: Funcionamiento OK", Color.YellowGreen);

    }

    if (bitArrEstado[4] == true)
    {
        this.lblEstadoMotorCilindro.Text = "ON";
        this.AddTextoLog("Estado motor cilindro: ON", Color.White);

    }
    else
    {
        this.lblEstadoMotorCilindro.Text = "OFF";
        this.AddTextoLog("Estado motor cilindro: OFF", Color.White);
    }

    if (bitArrEstado[5] == true)
    {
        this.lblPosicionCilindro.Text = "Extendido";
        this.AddTextoLog("Posición cilindro: Extendido", Color.White);
    }
    else
    {
        this.lblPosicionCilindro.Text = "Recogido";
        this.AddTextoLog("Posición cilindro: Recogido", Color.White);
    }

    if (bitArrEstado[6] == true)
    {
        this.lblElectroiman.Text = "ON";
        this.AddTextoLog("Electroimán: ON", Color.White);
    }
```

```
            else
            {
                this.lblElectroiman.Text = "OFF";
                this.AddTextoLog("Electroimán: OFF", Color.White);
            }

            if (bitArrEstado[7] == true)
            {
                this.lblEstadoTurbina.Text = "Girando";
                this.AddTextoLog("Estado turbina: Girando", Color.White);
            }
            else
            {
                this.lblEstadoTurbina.Text = "Parada";
                this.AddTextoLog("Estado turbina: Parada", Color.White);
            }

            //this.SetDesktopLocation(466, 0);
            this.Show();
            this.BringToFront();
        }

        private void InterpretaEstado(byte byEstado)
        {
            if (this.InvokeRequired == true)
            {
                Invoke(new delInterpretarEstado(this.Invoke_InterpretaEstado), new object[]
{ byEstado});
            }
            else
            {
                Invoke_InterpretaEstado(byEstado);
            }
        }
        #endregion

        private void AddTextoLog(string sTexto, Color color)
        {
            if (lvLogEstados.Items.Count >= 200)
            {//El log muestra 200 elementos como máximo
                lvLogEstados.Items.RemoveAt(199);
            }

            ListViewItem lvItem = new ListViewItem();

            lvItem.Text = DateTime.Now.ToString("dd/mm/yyyy HH:mm:ss.fff");
            lvItem.SubItems.Add(sTexto);
            lvItem.SubItems[1].BackColor = color;
            lvItem.UseItemStyleForSubItems = false;
            lvLogEstados.Items.Insert(0, lvItem); //Inserta elementos en el log siempre en
la primera línea desplazando el resto hacia abajo
            this.AgregarLineaLog(lvItem.Text + "\t|||\t" + lvItem.SubItems[1].Text);
//Agrega una nueva línea al log.txt
        }

        private void AgregarLineaLog(string sNuevaLinea)
```

```
        {
            try
            {
                streamWriter = new StreamWriter(@Properties.Settings.Default.rutaLogEstados,
true);
                streamWriter.WriteLine(sNuevaLinea);
                streamWriter.Close();
            }
            catch (Exception ex)
            {
            }
        }

        #endregion
    }
}
```

2.2.6 FormComandos.cs

```
/************************
 ** FormComandos.cs **
    31/05/2016
 ************************/
using System;
using System.Windows.Forms;

namespace ClienteGraficas
{
    public partial class FormComandos : Form
    {
        private TServicioUI servicio;

        #region Métodos
        public FormComandos()
        {
            InitializeComponent();
            servicio = TServicioUI.GetInstancia();
        }

        #region Eventos del formulario
        private void FormComandos_Load(object sender, EventArgs e)
        {
            this.Icon = Properties.Resources.icono;
        }

        //Con cada botón se envía el comando correspondiente vía TCP. Sí se trata de un
comando con consigna también se envía
        private void btnArrancarTurbina_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0xAA, 0);
        }
```

```
        private void btnPararTurbina_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0xFF, 0);
        }

        private void btnSendConsgVel_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0x0A, (Int16)spEditConsignaVel.Value);
        }

        private void btnFuncionamientoNormal_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0x00, 0);
        }

        private void btnAbrirValvula_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0xF0, 0);
        }

        private void btnCerrarValvula_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0x0F, 0);
        }

        private void btnSendConsgValvula_Click(object sender, EventArgs e)
        {
            servicio.EnvioDeComando(0xA0, (Int16)spEditConsignaValvula.Value);
        }

        private void FormComandos_FormClosing(object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            Hide();
        }
        #endregion
        #endregion
    }
}
```

2.2.7 FormConfigCurvas.cs

```
/**************************
 ** FormConfigCurvas.cs **
    31/05/2016
 **************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using Microsoft.Win32;
using DevExpress.XtraCharts;

namespace ClienteGraficas
{
    public partial class FormConfigCurvas : Form
    {
        public TConfigCurvas configCurvas;

        public delegate void NuevaConfigCurvasEventHandler(TConfigCurvas config);
        public event NuevaConfigCurvasEventHandler eventoNuevaConfigCurvas;

        #region Métodos
        public FormConfigCurvas()
        {
        }

        public FormConfigCurvas(TConfigCurvas curv)
        {
            InitializeComponent();
            this.configCurvas = curv;
        }

        #region Eventos del formulario
        private void FormConfigCurvas_Load(object sender, EventArgs e)
        {
            this.Icon = Properties.Resources.icono;

            this.ActualizarCamposFormulario();
        }

        private void btnAceptar_Click(object sender, EventArgs e)
        {//Guarda los valores modificados del formulario en el registro
            configCurvas.serieVelGenerador.View.Color = this.cPickVelGen.Color;
            configCurvas.seriePresCamara.View.Color = this.cPickPres.Color;
            configCurvas.seriePosCilindro.View.Color = this.cPickPosCil.Color;
            configCurvas.seriePotencia.View.Color = this.cPickPot.Color;
            configCurvas.serieVibraciones.View.Color = this.cPickVibraciones.Color;
            configCurvas.serieCorrienteIU.View.Color = this.cPickCorrIU.Color;
            configCurvas.serieCorrienteIV.View.Color = this.cPickCorrIV.Color;
            configCurvas.serieTensionVDC.View.Color = this.cPickTens.Color;
            configCurvas.serieTemp1.View.Color = this.cPickTemp1.Color;
            configCurvas.serieTemp2.View.Color = this.cPickTemp2.Color;
            configCurvas.serieComPosValvula.View.Color = this.cPickComPos.Color;
            configCurvas.serieRefVelMax.View.Color = this.cPickRefVelMax.Color;

            configCurvas.confEjeVel.minimoY = float.Parse(this.txtEditMinVel.Text);
            configCurvas.confEjeVel.maximoY = float.Parse(this.txtEditMaxVel.Text);
            configCurvas.confEjePres.minimoY = float.Parse(this.txtEditMinPres.Text);
            configCurvas.confEjePres.maximoY = float.Parse(this.txtEditMaxPres.Text);
            configCurvas.confEjePot.minimoY = float.Parse(this.txtEditMinPot.Text);
            configCurvas.confEjePot.maximoY = float.Parse(this.txtEditMaxPot.Text);
```

```
            configCurvas.confEjeVibr.minimoY = float.Parse(this.txtEditMinVibr.Text);
            configCurvas.confEjeVibr.maximoY = float.Parse(this.txtEditMaxVibr.Text);
            configCurvas.confEjePosCil.minimoY = float.Parse(this.txtEditMinPosCil.Text);
            configCurvas.confEjePosCil.maximoY = float.Parse(this.txtEditMaxPosCil.Text);
            configCurvas.confEjeCorr.minimoY = float.Parse(this.txtEditMinCorr.Text);
            configCurvas.confEjeCorr.maximoY = float.Parse(this.txtEditMaxCorr.Text);
            configCurvas.confEjeTens.minimoY = float.Parse(this.txtEditMinTens.Text);
            configCurvas.confEjeTens.maximoY = float.Parse(this.txtEditMaxTens.Text);
            configCurvas.confEjeTemp.minimoY = float.Parse(this.txtEditMinTemp.Text);
            configCurvas.confEjeTemp.maximoY = float.Parse(this.txtEditMaxTemp.Text);
            configCurvas.confEjeComPos.minimoY = float.Parse(this.txtEditMinComPos.Text);
            configCurvas.confEjeComPos.maximoY = float.Parse(this.txtEditMaxComPos.Text);

            if(this.eventoNuevaConfigCurvas!=null)
            {
                this.eventoNuevaConfigCurvas(configCurvas);
            }

            this.ActualizarRegistro();
            this.Close();
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {//Salir sin guardar los cambios realizados
            this.Close();
        }
        #endregion

        private void ActualizarCamposFormulario()
        {//Muestra en el formulario los valores guardados en el registro de la configuración
de curvas
            this.cPickVelGen.Color = configCurvas.serieVelGenerador.View.Color;
            this.cPickPres.Color = configCurvas.seriePresCamara.View.Color;
            this.cPickPosCil.Color = configCurvas.seriePosCilindro.View.Color;
            this.cPickPot.Color = configCurvas.seriePotencia.View.Color;
            this.cPickVibraciones.Color = configCurvas.serieVibraciones.View.Color;
            this.cPickCorrIU.Color = configCurvas.serieCorrienteIU.View.Color;
            this.cPickCorrIV.Color = configCurvas.serieCorrienteIV.View.Color;
            this.cPickTens.Color = configCurvas.serieTensionVDC.View.Color;
            this.cPickTemp1.Color = configCurvas.serieTemp1.View.Color;
            this.cPickTemp2.Color = configCurvas.serieTemp2.View.Color;
            this.cPickComPos.Color = configCurvas.serieComPosValvula.View.Color;
            this.cPickRefVelMax.Color = configCurvas.serieRefVelMax.View.Color;

            this.txtEditMinVel.Text = configCurvas.confEjeVel.minimoY.ToString();
            this.txtEditMaxVel.Text = configCurvas.confEjeVel.maximoY.ToString();
            this.txtEditMinPres.Text = configCurvas.confEjePres.minimoY.ToString();
            this.txtEditMaxPres.Text = configCurvas.confEjePres.maximoY.ToString();
            this.txtEditMinPot.Text = configCurvas.confEjePot.minimoY.ToString();
            this.txtEditMaxPot.Text = configCurvas.confEjePot.maximoY.ToString();
            this.txtEditMinVibr.Text = configCurvas.confEjeVibr.minimoY.ToString();
            this.txtEditMaxVibr.Text = configCurvas.confEjeVibr.maximoY.ToString();
            this.txtEditMinCorr.Text = configCurvas.confEjeCorr.minimoY.ToString();
            this.txtEditMaxCorr.Text = configCurvas.confEjeCorr.maximoY.ToString();
            this.txtEditMinTens.Text = configCurvas.confEjeTens.minimoY.ToString();
            this.txtEditMaxTens.Text = configCurvas.confEjeTens.maximoY.ToString();
```

```
            this.txtEditMinTemp.Text = configCurvas.confEjeTemp.minimoY.ToString();
            this.txtEditMaxTemp.Text = configCurvas.confEjeTemp.maximoY.ToString();
            this.txtEditMinComPos.Text = configCurvas.confEjeComPos.minimoY.ToString();
            this.txtEditMaxComPos.Text = configCurvas.confEjeComPos.maximoY.ToString();
            this.txtEditMinPosCil.Text = configCurvas.confEjePosCil.minimoY.ToString();
            this.txtEditMaxPosCil.Text = configCurvas.confEjePosCil.maximoY.ToString();
        }

        #region Actualiza registro
        private void ActualizarRegistro()
        {//Abre el registro de ITRESA y el subregistro correspondiente a la aplicación de la
turbina
            try
            {
                RegistryKey regKeySoft = Registry.CurrentUser.OpenSubKey("Software", true);
                RegistryKey regKeyITRESA = regKeySoft.OpenSubKey("ITRESA", true);
                RegistryKey regKeyApp = regKeyITRESA.OpenSubKey("App_SCADA_TURBINA", true);

                this.ActualizarRegistroEjes(regKeyApp);
                this.ActualizarRegistroSeries(regKeyApp);

                if (regKeySoft != null)
                {
                    regKeySoft.Close();
                }
                if (regKeyITRESA != null)
                {
                    regKeyITRESA.Close();
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show("No se ha podido actualizar el registro: " + ex.Message,
"Advertencia", MessageBoxButtons.OK);
            }
        }

        private void ActualizarRegistroEjes(RegistryKey regApp)
        {
            this.ActualizarRegEje("EjeVelocidades", regApp, ref configCurvas.confEjeVel);
            this.ActualizarRegEje("EjePresion", regApp, ref configCurvas.confEjePres);
            this.ActualizarRegEje("EjePosicionCilindro", regApp, ref
configCurvas.confEjePosCil);
            this.ActualizarRegEje("EjePotencia", regApp, ref configCurvas.confEjePot);
            this.ActualizarRegEje("EjeVibraciones", regApp, ref configCurvas.confEjeVibr);
            this.ActualizarRegEje("EjeCorrientes", regApp, ref configCurvas.confEjeCorr);
            this.ActualizarRegEje("EjeTensionVDC", regApp, ref configCurvas.confEjeTens);
            this.ActualizarRegEje("EjeTemperaturas", regApp, ref configCurvas.confEjeTemp);
            this.ActualizarRegEje("EjeComandoPosValvula", regApp, ref
configCurvas.confEjeComPos);
        }

        private void ActualizarRegEje(string sNombreEje, RegistryKey regApp, ref TConfEjeY
confEje)
        {//Abre el registro del eje correspondiente para actualizarlo
            RegistryKey regEje = null;
```

```
            try
            {
                regEje = regApp.OpenSubKey(sNombreEje, true);
                if (regEje != null)
                {
                    regEje.SetValue("Minimo", confEje.minimoY);
                    regEje.SetValue("Maximo", confEje.maximoY);
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show("No se ha podido actualizar el registro: " + ex.Message,
"Advertencia", MessageBoxButtons.OK);
            }
        }

        private void ActualizarRegistroSeries(RegistryKey regApp)
        {
            this.ActualizarRegSerie("SerieVelocidadGenerador", regApp,
configCurvas.serieVelGenerador);
            this.ActualizarRegSerie("SeriePresionCamara", regApp,
configCurvas.seriePresCamara);
            this.ActualizarRegSerie("SeriePosicionCilindro", regApp,
configCurvas.seriePosCilindro);
            this.ActualizarRegSerie("SeriePotencia", regApp, configCurvas.seriePotencia);
            this.ActualizarRegSerie("SerieVibraciones", regApp,
configCurvas.serieVibraciones);
            this.ActualizarRegSerie("SerieCorrienteIU", regApp,
configCurvas.serieCorrienteIU);
            this.ActualizarRegSerie("SerieCorrienteIV", regApp,
configCurvas.serieCorrienteIV);
            this.ActualizarRegSerie("SerieTensionVDC", regApp,
configCurvas.serieTensionVDC);
            this.ActualizarRegSerie("SerieTemp1", regApp, configCurvas.serieTemp1);
            this.ActualizarRegSerie("SerieTemp2", regApp, configCurvas.serieTemp2);
            this.ActualizarRegSerie("SerieComandoPosValvula", regApp,
configCurvas.serieComPosValvula);
            this.ActualizarRegSerie("SerieRefVelGiroMax", regApp,
configCurvas.serieRefVelMax);
        }

        private void ActualizarRegSerie(string sNombreSerie, RegistryKey regApp, Series
serieRegistro)
        {//Abre el registro de la serie correspondiente para actualizarla
            RegistryKey regSerie = null;

            try
            {
                regSerie = regApp.OpenSubKey(sNombreSerie, true);
                if (regSerie != null)
                {
                    regSerie.SetValue("Visible", serieRegistro.Visible);
                    regSerie.SetValue("Color", serieRegistro.View.Color.ToArgb());
                }
            }
            catch(Exception ex)
```

```
            {
                MessageBox.Show("No se ha podido actualizar el registro: " + ex.Message,
"Advertencia", MessageBoxButtons.OK);
            }
        }
        #endregion
        #endregion
    }
}
```

2.2.8 FormExportando.cs

```
/************************
 ** FormExportando.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;

namespace ClienteGraficas
{
    public partial class FormExportando : Form
    {
        #region Métodos
        public FormExportando()
        {
            InitializeComponent();
        }

        #region Eventos del formulario
        private void FormExportando_FormClosed(object sender, FormClosedEventArgs e)
        {//Escoje entre los procesos activos de la lista de tareas de Windows y "mata" los
relacionados con Excel
            foreach (Process proceso in Process.GetProcesses())
            {
                if (proceso.ProcessName == "EXCEL")
                {
                    proceso.Kill();
                }
            }
        }

        private void FormExportando_Load(object sender, EventArgs e)
        {
```

```
            this.Icon = Properties.Resources.icono;
        }
        #endregion
        #endregion
    }
}
```

2.2.9 FormNuevoGrafico.cs

```
/**************************
 ** FormNuevoGrafico.cs **
    31/05/2016
 **************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Threading;
using Microsoft.Win32;
using DevExpress.XtraCharts;
using System.Data.SqlClient;

namespace ClienteGraficas
{
    public partial class FormNuevoGrafico : Form
    {
        private TConfigCurvas configCurvas;
        private DateTime dtFechaDesde;
        private DateTime dtFechaHasta;
        private XYDiagram diagram;
        private TExportaExcel exportaExcel;
        private Thread hiloExcel;
        private string rutaSave;

        #region Métodos
        public FormNuevoGrafico(TConfigCurvas curv)
        {//Constructor para los gráficos no predefinidos.
            InitializeComponent();
            configCurvas = curv;
            chckListCurvas.SetItemCheckState(0, CheckState.Checked);
            chckListCurvas.SetItemCheckState(1, CheckState.Checked);
            chckListCurvas.SetItemCheckState(3, CheckState.Checked);
            chckListCurvas.SetItemCheckState(8, CheckState.Checked);
            diagram = chartCurvas.Diagram as XYDiagram;
            this.SetSeriesVisibility();
        }
```

```
        public FormNuevoGrafico(TConfigCurvas curv, string[] sCurvasPredet)
        {//Constructor para los gráficos predefinidos.
            InitializeComponent();
            configCurvas = curv;
            foreach (string nombre in sCurvasPredet)
            {
                for (int index = 0; index < this.chckListCurvas.Items.Count; index++)
                {
                    if (chckListCurvas.Items[index].Description == nombre)
                    {
                        chckListCurvas.SetItemCheckState(index, CheckState.Checked);
                    }
                }
            }
            diagram = chartCurvas.Diagram as XYDiagram;
            this.SetSeriesVisibility();
            chckListCurvas.Enabled = false;
        }

        #region Eventos del formulario
        private void FormNuevoGrafico_Load(object sender, EventArgs e)
        {
            this.Icon = Properties.Resources.icono;
            this.ConfiguraEjesySeries();
               this.EstablecerNombreForm(); //Modifica el nombre de la ventana
        }

        private void dtEditDesde_EditValueChanged(object sender, EventArgs e)
        {
            dtEditHasta.Properties.MinValue = dtEditDesde.DateTime;
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            if (this.SetSeriesVisibility() == true)
            {//Si hay alguna curva seleccionada para mostrar
                this.EstablecerNombreForm(); //Modifica el nombre de la ventana
                try
                {
                    dtFechaDesde = DateTime.Parse(dtEditDesde.Text + " " + tEditDesde.Text);
                    dtFechaHasta = DateTime.Parse(dtEditHasta.Text + " " + tEditHasta.Text);
                    //Actualiza el gráfico con los datos resultado de la consulta a la base
de datos
                    dSetDatosTurbinaLocal.tablaGraficas.Clear();

tablaGraficasTableAdapter.FillByFechas(dSetDatosTurbinaLocal.tablaGraficas, dtFechaDesde,
dtFechaHasta);
                    this.chartCurvas.Titles[0].Text = "Desde " +
dtFechaDesde.ToString("dd/MM/yyyy HH:mm:ss.fff") +
                                                      " Hasta " +
dtFechaHasta.ToString("dd/MM/yyyy HH:mm:ss.fff");
                    chartCurvas.Update();
                }
                catch (Exception ex)
                {
```

```
                    MessageBox.Show("Error al actualizar el gráfico con datos de la BBDD. "
+ ex.Message);
                }
            }
        }

        private void btnExportarExcel_Click(object sender, EventArgs e)
        {//Botón iniciar exportación a excel pulsado
            if (chartCurvas.Diagram != null)
            {//Si hay algún dato mostrado en el gráfico
                if (DialogResult.OK == MessageBox.Show("Se exportarán los datos de las
curvas:\n\t" + this.Text +
                    "\nDesde: " + dtFechaDesde.ToString() +
                    "\nHasta:  " + dtFechaHasta.ToString(), "Exportar",
MessageBoxButtons.OKCancel)
                  )
                {
                    DialogResult dialogResult = examinarDialog.ShowDialog();
                    rutaSave = examinarDialog.SelectedPath;
                    if(DialogResult.OK == dialogResult)
                    {
                        if (rutaSave != "")
                        {//Si se ha seleccionado la ruta y aceptado la exportación
                            if (dSetDatosTurbinaLocal.tablaGraficas.Rows.Count > 0)
                            {
                                hiloExcel = new Thread(new ThreadStart(HiloExcel));
                                hiloExcel.Start();
                            }

                            else
                            {
                                MessageBox.Show("No hay datos para esas fechas", "Aviso",
MessageBoxButtons.OK);
                            }
                        }
                    }
                }
            }
            else
            {
                MessageBox.Show("Muestre un gráfico para exportar", "Aviso",
MessageBoxButtons.OK);
            }

        }

        private void ZoomInButton_Click(object sender, EventArgs e)
        {//Botón Zoom In pulsado modifica la escala del eje X
            if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Year)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Month;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Month;
                lblZoom.Text = "Mes";
```

```
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy}";

            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Month)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Day;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Day;
                lblZoom.Text = "Día";
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Day)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Hour;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Hour;
                lblZoom.Text = "Hora";
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy HH:mm}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Hour)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Minute;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Minute;
                lblZoom.Text = "Minuto";
                diagram.AxisX.Label.TextPattern = "{A:HH:mm:ss}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Minute)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Second;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Second;
                lblZoom.Text = "Segundo";
                diagram.AxisX.Label.TextPattern = "{A:HH:mm:ss}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Second)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Millisecond;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Millisecond;
                lblZoom.Text = "Milisegundo";
                diagram.AxisX.Label.TextPattern = "{A:HH:mm:ss.fff}";
            }
        }

        private void ZoomOutButton_Click(object sender, EventArgs e)
```

```
        {//Botón Zoom Out pulsado modifica la escala del eje X
            if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Millisecond)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Second;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Second;
                lblZoom.Text = "Segundo";
                diagram.AxisX.Label.TextPattern = "{A:HH:mm:ss}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Second)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Minute;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Minute;
                lblZoom.Text = "Minuto";
                diagram.AxisX.Label.TextPattern = "{A:HH:mm:ss}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Minute)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Hour;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Hour;
                lblZoom.Text = "Hora";
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy HH:mm}";

            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Hour)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Day;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Day;
                lblZoom.Text = "Día";
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Day)
            {
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Month;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Month;
                lblZoom.Text = "Mes";
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy}";
            }
            else if (diagram.AxisX.DateTimeScaleOptions.MeasureUnit ==
DevExpress.XtraCharts.DateTimeMeasureUnit.Month)
            {
```

```
                diagram.AxisX.DateTimeScaleOptions.MeasureUnit =
DevExpress.XtraCharts.DateTimeMeasureUnit.Year;
                diagram.AxisX.DateTimeScaleOptions.GridAlignment =
DevExpress.XtraCharts.DateTimeGridAlignment.Year;
                lblZoom.Text = "Año";
                diagram.AxisX.Label.TextPattern = "{A:dd/MM/yy}";
            }
        }

        private void FormNuevoGrafico_FormClosed(object sender, FormClosedEventArgs e)
        {

        }
        #endregion

        private bool SetSeriesVisibility()
        {//comprueba los elementos marcados en el checkList
            if(chckListCurvas.Items["Velocidad generador"].CheckState==CheckState.Checked)
            {
                chartCurvas.Series["Velocidad generador"].Visible = true;
            }
            else
            {
                chartCurvas.Series["Velocidad generador"].Visible = false;
            }
            if (chckListCurvas.Items["Presión cámara"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Presión cámara"].Visible = true;
                diagram.SecondaryAxesY[1].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                chartCurvas.Series["Presión cámara"].Visible = false;
                diagram.SecondaryAxesY[1].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Posición cilindro"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Posición cilindro"].Visible = true;
                diagram.SecondaryAxesY[2].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                chartCurvas.Series["Posición cilindro"].Visible = false;
                diagram.SecondaryAxesY[2].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Potencia"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Potencia"].Visible = true;
                diagram.SecondaryAxesY[3].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                chartCurvas.Series["Potencia"].Visible = false;
```

```
                diagram.SecondaryAxesY[3].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Vibraciones"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Vibraciones"].Visible = true;
                diagram.SecondaryAxesY[4].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                chartCurvas.Series["Vibraciones"].Visible = false;
                diagram.SecondaryAxesY[4].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Corriente IU"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Corriente IU"].Visible = true;
            }
            else
            {
                chartCurvas.Series["Corriente IU"].Visible = false;
            }
            if (chckListCurvas.Items["Corriente IV"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Corriente IV"].Visible = true;
            }
            else
            {
                chartCurvas.Series["Corriente IV"].Visible = false;
            }
            if (chckListCurvas.Items["Tensión VDC"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Tensión VDC"].Visible = true;
                diagram.SecondaryAxesY[6].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                chartCurvas.Series["Tensión VDC"].Visible = false;
                diagram.SecondaryAxesY[6].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Temperatura 1"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Temperatura 1"].Visible = true;
            }
            else
            {
                chartCurvas.Series["Temperatura 1"].Visible = false;
            }
            if (chckListCurvas.Items["Temperatura 2"].CheckState == CheckState.Checked)
            {
                chartCurvas.Series["Temperatura 2"].Visible = true;
            }
            else
            {
                chartCurvas.Series["Temperatura 2"].Visible = false;
```

```
            }
            if (chckListCurvas.Items["Comando posición válvula"].CheckState ==
CheckState.Checked)
            {
                chartCurvas.Series["Comando posición válvula"].Visible = true;
                diagram.SecondaryAxesY[8].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                chartCurvas.Series["Comando posición válvula"].Visible = false;
                diagram.SecondaryAxesY[8].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Referencia velocidad giro máx."].CheckState ==
CheckState.Checked)
            {
                chartCurvas.Series["Referencia velocidad giro máx."].Visible = true;
            }
            else
            {
                chartCurvas.Series["Referencia velocidad giro máx."].Visible = false;
            }

            if (chckListCurvas.Items["Velocidad generador"].CheckState == CheckState.Checked
|| chckListCurvas.Items["Referencia velocidad giro máx."].CheckState == CheckState.Checked)
            {
                diagram.SecondaryAxesY[0].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                diagram.SecondaryAxesY[0].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Corriente IU"].CheckState == CheckState.Checked ||
chckListCurvas.Items["Corriente IV"].CheckState == CheckState.Checked)
            {
                diagram.SecondaryAxesY[5].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                diagram.SecondaryAxesY[5].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.Items["Temperatura 1"].CheckState == CheckState.Checked ||
chckListCurvas.Items["Temperatura 2"].CheckState == CheckState.Checked)
            {
                diagram.SecondaryAxesY[7].Visibility = DevExpress.Utils.DefaultBoolean.True;
            }
            else
            {
                diagram.SecondaryAxesY[7].Visibility =
DevExpress.Utils.DefaultBoolean.False;
            }
            if (chckListCurvas.CheckedItems.Count == 0)
            {//si no hay ningún elemento marcado en el checkList
```

```
                MessageBox.Show("Seleccione al menos una curva para mostrar", "Error",
MessageBoxButtons.OK);
                return false;
            }
            else
            {
                return true;
            }
        }

        private void ConfiguraEjesySeries()
        {
            try
            {
                //Eje velocidad
                diagram.SecondaryAxesY[0].WholeRange.MaxValue =
configCurvas.confEjeVel.maximoY;
                diagram.SecondaryAxesY[0].WholeRange.MinValue =
configCurvas.confEjeVel.minimoY;
                diagram.SecondaryAxesY[0].VisualRange.MaxValue =
configCurvas.confEjeVel.maximoY;
                diagram.SecondaryAxesY[0].VisualRange.MinValue =
configCurvas.confEjeVel.minimoY;
                //Eje presión
                diagram.SecondaryAxesY[1].WholeRange.MaxValue =
configCurvas.confEjePres.maximoY;
                diagram.SecondaryAxesY[1].WholeRange.MinValue =
configCurvas.confEjePres.minimoY;
                diagram.SecondaryAxesY[1].VisualRange.MaxValue =
configCurvas.confEjePres.maximoY;
                diagram.SecondaryAxesY[1].VisualRange.MinValue =
configCurvas.confEjePres.minimoY;
                //Eje posición cilindro
                diagram.SecondaryAxesY[2].WholeRange.MaxValue =
configCurvas.confEjePosCil.maximoY;
                diagram.SecondaryAxesY[2].WholeRange.MinValue =
configCurvas.confEjePosCil.minimoY;
                diagram.SecondaryAxesY[2].VisualRange.MaxValue =
configCurvas.confEjePosCil.maximoY;
                diagram.SecondaryAxesY[2].VisualRange.MinValue =
configCurvas.confEjePosCil.minimoY;
                //Eje potencia
                diagram.SecondaryAxesY[3].WholeRange.MaxValue =
configCurvas.confEjePot.maximoY;
                diagram.SecondaryAxesY[3].WholeRange.MinValue =
configCurvas.confEjePot.minimoY;
                diagram.SecondaryAxesY[3].VisualRange.MaxValue =
configCurvas.confEjePot.maximoY;
                diagram.SecondaryAxesY[3].VisualRange.MinValue =
configCurvas.confEjePot.minimoY;
                //Eje vibraciones
                diagram.SecondaryAxesY[4].WholeRange.MaxValue =
configCurvas.confEjeVibr.maximoY;
                diagram.SecondaryAxesY[4].WholeRange.MinValue =
configCurvas.confEjeVibr.minimoY;
```

```
                diagram.SecondaryAxesY[4].VisualRange.MaxValue =
configCurvas.confEjeVibr.maximoY;
                diagram.SecondaryAxesY[4].VisualRange.MinValue =
configCurvas.confEjeVibr.minimoY;
                //Eje corriente
                diagram.SecondaryAxesY[5].WholeRange.MaxValue =
configCurvas.confEjeCorr.maximoY;
                diagram.SecondaryAxesY[5].WholeRange.MinValue =
configCurvas.confEjeCorr.minimoY;
                diagram.SecondaryAxesY[5].VisualRange.MaxValue =
configCurvas.confEjeCorr.maximoY;
                diagram.SecondaryAxesY[5].VisualRange.MinValue =
configCurvas.confEjeCorr.minimoY;
                //Eje tensión
                diagram.SecondaryAxesY[6].WholeRange.MaxValue =
configCurvas.confEjeTens.maximoY;
                diagram.SecondaryAxesY[6].WholeRange.MinValue =
configCurvas.confEjeTens.minimoY;
                diagram.SecondaryAxesY[6].VisualRange.MaxValue =
configCurvas.confEjeTens.maximoY;
                diagram.SecondaryAxesY[6].VisualRange.MinValue =
configCurvas.confEjeTens.minimoY;
                //Eje temperatura
                diagram.SecondaryAxesY[7].WholeRange.MaxValue =
configCurvas.confEjeTemp.maximoY;
                diagram.SecondaryAxesY[7].WholeRange.MinValue =
configCurvas.confEjeTemp.minimoY;
                diagram.SecondaryAxesY[7].VisualRange.MaxValue =
configCurvas.confEjeTemp.maximoY;
                diagram.SecondaryAxesY[7].VisualRange.MinValue =
configCurvas.confEjeTemp.minimoY;
                //Eje comando posición válvula
                diagram.SecondaryAxesY[8].WholeRange.MaxValue =
configCurvas.confEjeComPos.maximoY;
                diagram.SecondaryAxesY[8].WholeRange.MinValue =
configCurvas.confEjeComPos.minimoY;
                diagram.SecondaryAxesY[8].VisualRange.MaxValue =
configCurvas.confEjeComPos.maximoY;
                diagram.SecondaryAxesY[8].VisualRange.MinValue =
configCurvas.confEjeComPos.minimoY;

                //Colores series
                chartCurvas.Series["Velocidad generador"].View.Color =
configCurvas.serieVelGenerador.View.Color;
                chartCurvas.Series["Presión cámara"].View.Color =
configCurvas.seriePresCamara.View.Color;
                chartCurvas.Series["Posición cilindro"].View.Color =
configCurvas.seriePosCilindro.View.Color;
                chartCurvas.Series["Potencia"].View.Color =
configCurvas.seriePotencia.View.Color;
                chartCurvas.Series["Vibraciones"].View.Color =
configCurvas.serieVibraciones.View.Color;
                chartCurvas.Series["Corriente IU"].View.Color =
configCurvas.serieCorrienteIU.View.Color;
                chartCurvas.Series["Corriente IV"].View.Color =
configCurvas.serieCorrienteIV.View.Color;
```

```
                chartCurvas.Series["Tensión VDC"].View.Color =
configCurvas.serieTensionVDC.View.Color;
                chartCurvas.Series["Temperatura 1"].View.Color =
configCurvas.serieTemp1.View.Color;
                chartCurvas.Series["Temperatura 2"].View.Color =
configCurvas.serieTemp2.View.Color;
                chartCurvas.Series["Comando posición válvula"].View.Color =
configCurvas.serieComPosValvula.View.Color;
                chartCurvas.Series["Referencia velocidad giro máx."].View.Color =
configCurvas.serieRefVelMax.View.Color;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al leer del registro. " + ex.Message);
            }
        }

        private void EstablecerNombreForm()
        {//Modifica el nombre del formulario
            if (chckListCurvas.CheckedItemsCount > 0)
            {
                string sSeriesChecked = "";
                foreach (object itemChecked in chckListCurvas.CheckedItems)
                {
                    sSeriesChecked += "-" + itemChecked.ToString() + "-";
                }
                this.Text = sSeriesChecked;
            }
            else
            {
                this.Text = "Nuevo gráfico";
            }
        }

        public void HiloExcel()
        {//Inicia el hilo para la exportación a Excel
            exportaExcel = new TExportaExcel();
            exportaExcel.ExportarExcel(dSetDatosTurbinaLocal.tablaGraficas, chartCurvas,
dtFechaDesde, dtFechaHasta, chckListCurvas, rutaSave, configCurvas);
        }
        #endregion
    }
}
```

2.2.10 TClienteComandosTCP.cs

```
/*****************************
 ** TClienteComandosTCP.cs **
    31/05/2016
 *****************************/
using System;
using System.Threading;
```

```
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;

namespace ClienteGraficas
{
    public class TClienteComandosTCP
    {
        private int iPuertoTCP_Comandos; //Puerto en el que escucha el servidor de comandos

        private Int16 iReferencia; //Valor numérico de la referencia
        private byte byComando; //Comando codificado

        private Thread hiloClienteComandos; //El hilo se crea, se lanza y se termina con
cada envío

        #region Métodos
        public TClienteComandosTCP(byte comando, Int16 referencia)
        {
            iPuertoTCP_Comandos = Properties.Settings.Default.iPuertoTCP_Comandos;
            byComando = comando;
            iReferencia = referencia;

            hiloClienteComandos = new Thread(new ThreadStart(this.EnviaComando));
            hiloClienteComandos.Start();
        }

        private void EnviaComando()
        {
            try
            {
                IPAddress[] ipv4Direcciones =
Array.FindAll(Dns.GetHostEntry(Properties.Settings.Default.ipServidorComandos).AddressList,
                                                        d => d.AddressFamily ==
AddressFamily.InterNetwork);

                IPAddress ipAddress = ipv4Direcciones[0];

                IPEndPoint listener = new IPEndPoint(ipAddress, iPuertoTCP_Comandos);
//Parámetros del servidor de comandos
                Socket sender = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp); //Socket de envío

                sender.Connect(listener); //pide conexión
                //Si consigue conectar con el servidor envía los nuevos estados
                byte[] msg = this.ComponeTramaEnviar();
                sender.Send(msg);

                Thread.Sleep(1000); //retardo para el envío

                //Desconexión
                sender.Shutdown(SocketShutdown.Both);
                sender.Close();
            }
            catch (SocketException se)
```

```
            {//Si no es posible conectar con el servidor se genera una SocketException que
se mostrará en un MessageBox
                MessageBox.Show("Comando no enviado. Imposible conectar al servidor de
comandos." + "\n(" + se.ErrorCode + "). " + se.Message,
                                "Error de conexión", MessageBoxButtons.OK);
            }
        }

        private byte[] ComponeTramaEnviar()
        {
            byte[] byTramaTcpEnviar = new byte[7];
            byte[] byReferencia = new byte[2];
            byReferencia = BitConverter.GetBytes(iReferencia);

            byTramaTcpEnviar[0] = 0x10;
            byTramaTcpEnviar[1] = 0x02;
            byTramaTcpEnviar[2] = byComando;
            byTramaTcpEnviar[3] = byReferencia[1];
            byTramaTcpEnviar[4] = byReferencia[0];
            byTramaTcpEnviar[5] = 0x10;
            byTramaTcpEnviar[6] = 0x03;

            return byTramaTcpEnviar;
        }
        #endregion
    }
}
```

2.2.11 TConfigCurvas.cs

```
/************************
 ** TConfigCurvas.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using DevExpress.XtraCharts;

namespace ClienteGraficas
{
    public class TConfigCurvas
    {
        //Series
        public Series serieVelGenerador = new Series("Velocidad generador", ViewType.Line);
        public Series seriePresCamara = new Series("Presión cámara", ViewType.Line);
        public Series seriePosCilindro = new Series("Posición cilindro", ViewType.Line);
        public Series seriePotencia = new Series("Potencia", ViewType.Line);
        public Series serieVibraciones = new Series("Vibraciones", ViewType.Line);
```

```
        public Series serieCorrienteIU = new Series("Corriente IU", ViewType.Line);
        public Series serieCorrienteIV = new Series("Corriente IV", ViewType.Line);
        public Series serieTensionVDC = new Series("Tensión VDC", ViewType.Line);
        public Series serieTemp1 = new Series("Temperatura 1", ViewType.Line);
        public Series serieTemp2 = new Series("Temperatura 2", ViewType.Line);
        public Series serieComPosValvula = new Series("Comando posición válvula",
ViewType.Line);
        public Series serieRefVelMax = new Series("Referencia velocidad giro máx.",
ViewType.Line);

        //Configuración ejes
        public TConfEjeY confEjeVel = new TConfEjeY();
        public TConfEjeY confEjePres = new TConfEjeY();
        public TConfEjeY confEjePot = new TConfEjeY();
        public TConfEjeY confEjeVibr = new TConfEjeY();
        public TConfEjeY confEjePosCil = new TConfEjeY();
        public TConfEjeY confEjeCorr = new TConfEjeY();
        public TConfEjeY confEjeTens = new TConfEjeY();
        public TConfEjeY confEjeTemp = new TConfEjeY();
        public TConfEjeY confEjeComPos = new TConfEjeY();

        #region Métodos
        public TConfigCurvas()
        {
            //Asigna colores por defecto a las series
            this.serieVelGenerador.View.Color = System.Drawing.Color.Red;
            this.seriePresCamara.View.Color = System.Drawing.Color.Cyan;
            this.seriePosCilindro.View.Color = System.Drawing.Color.Green;
            this.seriePotencia.View.Color = System.Drawing.Color.Blue;
            this.serieVibraciones.View.Color = System.Drawing.Color.Black;
            this.serieCorrienteIU.View.Color = System.Drawing.Color.Gray;
            this.serieCorrienteIV.View.Color = System.Drawing.Color.Violet;
            this.serieTensionVDC.View.Color = System.Drawing.Color.YellowGreen;
            this.serieTemp1.View.Color = System.Drawing.Color.Fuchsia;
            this.serieTemp2.View.Color = System.Drawing.Color.Gold;
            this.serieComPosValvula.View.Color = System.Drawing.Color.Orange;
            this.serieRefVelMax.View.Color = System.Drawing.Color.Brown;
        }
        #endregion
    }

    public class TConfEjeY
    {
        //asigna rango por defecto a los ejes
        public float minimoY = 0;
        public float maximoY = 10000;
    }
}
```

2.2.12 TExportaExcel.cs

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

```
 ** TExportaExcel.cs **
    31/05/2016
 ************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Excel = Microsoft.Office.Interop.Excel;
using System.Windows.Forms;
using System.Data;
using DevExpress.XtraCharts;
using DevExpress.XtraEditors;

namespace ClienteGraficas
{
    class TExportaExcel
    {
        private FormExportando formExportando;

        #region Métodos
        public TExportaExcel()
        {
        }

        public void ExportarExcel(dSetDatosTurbinaLocal.tablaGraficasDataTable ds,
ChartControl ch, DateTime fDesde, DateTime fHasta, CheckedListBoxControl chck, string ruta,
TConfigCurvas cc)
        {
            Excel.Application excel;
            Excel.Workbook libroExcel;
            Excel.Worksheet hojaExcelTabla;
            Excel.Range rangoCeldasExcel;
            string sAhora = DateTime.Now.ToString().Replace('/', '-').Replace(':', '.');

            try
            {
                formExportando = new FormExportando();
                formExportando.Show();
                excel = new Excel.Application();
                excel.Visible = false;
                excel.DisplayAlerts = false;

                libroExcel = excel.Workbooks.Add(Type.Missing);

                hojaExcelTabla = (Excel.Worksheet)libroExcel.ActiveSheet;
                hojaExcelTabla.Name = "Datos";

                //cabecera de la hoja
                hojaExcelTabla.Cells[1, 1] = "Desde";
                hojaExcelTabla.Cells[1, 2] = fDesde.ToString();
                hojaExcelTabla.Cells[2, 1] = "Hasta";
                hojaExcelTabla.Cells[2, 2] = fDesde.ToString();
                hojaExcelTabla.Cells.Font.Color = System.Drawing.Color.Black;
                //Insertar los datos
```

```
                int filaExcel = 3;
                formExportando.prBar.Properties.Minimum = 0;
                formExportando.prBar.Properties.Maximum = ds.Rows.Count;
                formExportando.Text = "Creando tabla...";
                foreach (DataRow datarow in ds.Rows)
                {
                    if (filaExcel == 3)
                    {//nombres de columnas
                        hojaExcelTabla.Cells[3, 1] = ds.Columns["dtFecha"].ColumnName;
                        int columna = 2;
                        for (int i = 0; i < chck.Items.Count; i++)
                        {
                            if (chck.Items[i].CheckState == CheckState.Checked)
                            {
                                hojaExcelTabla.Cells[3, columna] =
ds.Columns[(string)chck.Items[i].Tag].ColumnName;
                                columna++;
                            }
                        }
                        filaExcel++;
                    }
                    if (filaExcel > 3)
                    {//Datos
                        for (int col = 1; col <= chck.CheckedItems.Count + 1; col++)
                        {
                            if (col == 1)
                            {
                                DateTime fecha = (DateTime)datarow[col - 1];//inserta fechas
                                hojaExcelTabla.Cells[filaExcel, col] =
fecha.ToString("dd/MM/yyyy HH:mm:ss.fff");
                            }
                            else
                            {
                                hojaExcelTabla.Cells[filaExcel, col] = datarow[col -
1].ToString(); //Inserta datos
                            }
                        }
                        filaExcel++;
                    }
                    formExportando.prBar.PerformStep();
                    formExportando.prBar.Update();
                }
                formExportando.prBar.EditValue = 0;
                formExportando.prBar.Update();

                //Formato celdas
                //de los datos
                rangoCeldasExcel = hojaExcelTabla.Range[hojaExcelTabla.Cells[3, 1],
hojaExcelTabla.Cells[filaExcel - 1, chck.CheckedItems.Count + 1]];
                rangoCeldasExcel.Font.Bold = false;
                Excel.Borders border = rangoCeldasExcel.Borders;
                border.LineStyle = Excel.XlLineStyle.xlContinuous;
                border.Weight = 2d;
                //de cabecera
                rangoCeldasExcel = hojaExcelTabla.Range[hojaExcelTabla.Cells[1, 1],
hojaExcelTabla.Cells[3, chck.CheckedItems.Count + 1]];
```

```
                rangoCeldasExcel.Font.Bold = true;
                //de tabla completa
                rangoCeldasExcel = hojaExcelTabla.Range[hojaExcelTabla.Cells[1, 1],
hojaExcelTabla.Cells[filaExcel - 1, chck.CheckedItems.Count + 1]];
                rangoCeldasExcel.EntireColumn.AutoFit();

                //Representa charts
                int iHojaAnterior = 1;
                formExportando.prBar.Properties.Minimum = 0;
                formExportando.prBar.Properties.Maximum = chck.Items.Count;
                formExportando.Text = "Creando gráficas...";
                for (int i = 0; i < chck.Items.Count; i++)
                {
                    if (chck.Items[i].CheckState == CheckState.Checked)
                    {
                        Excel.Worksheet hojaExcelChart;
                        libroExcel.Sheets.Add(libroExcel.Sheets[iHojaAnterior],
Type.Missing, Type.Missing, Type.Missing);
                        hojaExcelChart = (Excel.Worksheet)libroExcel.ActiveSheet;
                        hojaExcelChart.Name = (string)chck.Items[i].Value;

                        Excel.ChartObjects objCharts =
(Excel.ChartObjects)hojaExcelChart.ChartObjects(Type.Missing);
                        Excel.ChartObject chart = (Excel.ChartObject)objCharts.Add(0, 0,
800, 500);
                        Excel.Chart chartPage = chart.Chart;
                        Excel.SeriesCollection seriesCollection =
chartPage.SeriesCollection();

                        chartPage.Axes(Excel.XlAxisType.xlCategory,
Excel.XlAxisGroup.xlPrimary).TickLabelPosition =
Excel.XlTickLabelPosition.xlTickLabelPositionLow;
                        chartPage.ChartType = Excel.XlChartType.xlLine;
                        chartPage.ChartWizard(Title: String.Format("From {0} To {1}",
fDesde.ToString(), fHasta.ToString()));
                        chartPage.Axes(Excel.XlAxisType.xlCategory).Select();
                        chartPage.Axes(Excel.XlAxisType.xlCategory).TickLabels.Orientation =
-90;
                        chartPage.Axes(Excel.XlAxisType.xlCategory).TickLabelPosition =
Excel.XlTickLabelPosition.xlTickLabelPositionLow;
                        chartPage.Axes(Excel.XlAxisType.xlCategory).AxisBetweenCategories =
false;
                        chartPage.Axes(Excel.XlAxisType.xlCategory).HasMajorGridlines =
false;
                        chartPage.Axes(Excel.XlAxisType.xlCategory).HasMinorGridlines =
false;
                        chartPage.Axes(Excel.XlAxisType.xlValue).HasMinorGridlines = true;

                        TConfEjeY confEje = null;
                        Series serie = null;
                        if ((string)chck.Items[i].Value == "Velocidad generador")
                        {
                            confEje = cc.confEjeVel;
                            serie = cc.serieVelGenerador;
                        }
                        else if ((string)chck.Items[i].Value == "Presión cámara")
```

```
                {
                    confEje = cc.confEjePres;
                    serie = cc.seriePresCamara;
                }
                else if ((string)chck.Items[i].Value == "Posición cilindro")
                {
                    confEje = cc.confEjePosCil;
                    serie = cc.seriePosCilindro;
                }
                else if ((string)chck.Items[i].Value == "Potencia")
                {
                    confEje = cc.confEjePot;
                    serie = cc.seriePotencia;
                }
                else if ((string)chck.Items[i].Value == "Vibraciones")
                {
                    confEje = cc.confEjeVibr;
                    serie = cc.serieVibraciones;
                }
                else if ((string)chck.Items[i].Value == "Corriente IU")
                {
                    confEje = cc.confEjeCorr;
                    serie = cc.serieCorrienteIU;
                }
                else if ((string)chck.Items[i].Value == "Corriente IV")
                {
                    confEje = cc.confEjeCorr;
                    serie = cc.serieCorrienteIV;
                }
                else if ((string)chck.Items[i].Value == "Tensión VDC")
                {
                    confEje = cc.confEjeTens;
                    serie = cc.serieTensionVDC;
                }
                else if ((string)chck.Items[i].Value == "Temperatura 1")
                {
                    confEje = cc.confEjeTemp;
                    serie = cc.serieTemp1;
                }
                else if ((string)chck.Items[i].Value == "Temperatura 2")
                {
                    confEje = cc.confEjeTemp;
                    serie = cc.serieTemp2;
                }
                else if ((string)chck.Items[i].Value == "Comando posición válvula")
                {
                    confEje = cc.confEjeComPos;
                    serie = cc.serieComPosValvula;
                }
                else if ((string)chck.Items[i].Value == "Referencia velocidad giro
máx.")
                {
                    confEje = cc.confEjeVel;
                    serie = cc.serieRefVelMax;
                }
                //Rango ejes Y
```

```
                        chartPage.Axes(Excel.XlAxisType.xlValue).MinimumScale =
confEje.minimoY;
                        chartPage.Axes(Excel.XlAxisType.xlValue).MaximumScale =
confEje.maximoY;
                        //Serie
                        this.NuevaSerieExcelChart(
                                ds,
                                seriesCollection,
                                "dtFecha",
                                (string)chck.Items[i].Tag,
                                (string)chck.Items[i].Value,
                                serie
                            );

                        iHojaAnterior++;
                        formExportando.prBar.PerformStep();
                        formExportando.prBar.Update();
                    }
                }
                libroExcel.Sheets[1].Select();
                libroExcel.SaveAs(ruta + "\\" + sAhora + ".xlsx");
                libroExcel.Close();
                excel.Quit();
                excel = null;

                formExportando.prBar.EditValue = 0;
                formExportando.prBar.Update();
                formExportando.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error en la exportación a Excel. " + ex.Message);
            }
            finally
            {
                hojaExcelTabla = null;
                rangoCeldasExcel = null;
                libroExcel = null;
            }
        }

        private void NuevaSerieExcelChart(dSetDatosTurbinaLocal.tablaGraficasDataTable ds,
Excel.SeriesCollection seriesCollection, string xValues, string yValues, string nombreSerie,
Series serieColor)
        {
            Excel.Series serie = seriesCollection.NewSeries();
            int i = 0;
            object[] elementosX = new object[ds.Rows.Count];
            object[] elementosY = new object[ds.Rows.Count];
            foreach (DataRow fila in ds.Rows)
            {
                DateTime fecha = (DateTime)fila[xValues];
                elementosX[i] = fecha.ToString("HH:mm:ss.fff");
                elementosY[i] = fila[yValues];
                i++;
            }
```

```
            serie.XValues = elementosX;
            serie.Values = elementosY;
            serie.Name = nombreSerie;
            serie.Format.Line.ForeColor.RGB = this.ArgToRGB(serieColor);
        }

        private int ArgToRGB(Series serie)
        {
            int rojo = serie.View.Color.R;
            int verde = serie.View.Color.G;
            int azul = serie.View.Color.B;

            return (int)(rojo + (verde * 256) + (azul * 256 * 256));
        }
        #endregion
    }
}
```

2.2.13 TServidorEstadosTCP.cs

```
/*****************************
 ** TServidorEstadosTCP.cs **
    31/05/2016
 *****************************/
using System;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;

namespace ClienteGraficas
{
    class StateObject
    {
        public Socket listener = null;
        public const int BufferSize = 1024;
        public byte[] buffer = new byte[BufferSize];
        public byte[] byMensajeRecibido;
        public StateObject() { }
    }

    class TServidorEstadosTCP
    {//Escucha asincrónica de conexiones entrantes vía TCP/IP
        private ManualResetEvent clienteConectado;
        private ManualResetEvent recibido;

        private Thread hiloServidorEstados;

        private int iPuertoTCP_Estados; //número del puerto de escucha
        private int iLongitudTramaEstados;

        private bool bFinalizarHilo;
```

```
        public delegate void delMensajeRecibidoHandler(byte byError1, byte byError2, byte
byError3, byte byEstado);
        public event delMensajeRecibidoHandler eventoMensajeRecibido;

        #region Métodos
        public TServidorEstadosTCP()
        {
            clienteConectado = new ManualResetEvent(false);
            recibido = new ManualResetEvent(false);
            iPuertoTCP_Estados = Properties.Settings.Default.iPuertoTCP_Estados;
            iLongitudTramaEstados = Properties.Settings.Default.iLongTramaEstados;
            bFinalizarHilo = false;


        }

        public void IniciarHiloServidor()
        {
            hiloServidorEstados = new Thread(new ThreadStart(HiloEscuchar));
            hiloServidorEstados.Start();
        }

        private void HiloEscuchar()
        {
            IPEndPoint socket = new IPEndPoint(IPAddress.Any, iPuertoTCP_Estados);

            try
            {//Se espera que el cliente se conecte y desconecte del servidor con cada envío
de comandos
                using (Socket listener = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp))
                {
                    listener.Bind(socket);
                    listener.Listen(1);
                    while (bFinalizarHilo == false)
                    {//Espera por conexión del cliente. Si se desconecta vuelve a esperar
por otra conexión
                        clienteConectado.Reset();
                        listener.BeginAccept(new
AsyncCallback(this.ClienteConectadoCallback), listener);
                        clienteConectado.WaitOne(); //bloqueo
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se puede conectar con el Cliente de estados: " +
ex.Message, "Advertencia", MessageBoxButtons.OK);
            }
        }

        private void ClienteConectadoCallback(IAsyncResult result)
        {
            clienteConectado.Set(); //cliente conectado, desbloqueo
            StateObject st = new StateObject();
```

```
            try
            {
                st.listener = (Socket)result.AsyncState;
                st.listener = st.listener.EndAccept(result);
                //Espera a que termine la recepción de comandos
                recibido.Reset();
                st.listener.BeginReceive(st.buffer, 0, StateObject.BufferSize,
SocketFlags.None, new AsyncCallback(this.RecibiendoCallback), st);
                recibido.WaitOne(); //bloqueo
            }
            catch (Exception ex)
            {
            }
        }

        private void RecibiendoCallback(IAsyncResult result)
        {
            StateObject st = (StateObject)result.AsyncState;
            try
            {
                recibido.Set(); //recepción terminada, desbloqueo
                int bytesRecibidos = st.listener.EndReceive(result);
                st.byMensajeRecibido = new byte[bytesRecibidos];

                if (bytesRecibidos == iLongitudTramaEstados)
                {//Extracción de los bytes del comando de la trama completa (de 1024 bytes)
                    Buffer.BlockCopy(st.buffer, 0, st.byMensajeRecibido, 0, bytesRecibidos);

                    if (eventoMensajeRecibido != null)
                    {//en el byte 2 está el tipo de comando y en el 3 y 4 las partes alta y
baja de la referencia cuando corresponda
                        eventoMensajeRecibido(st.byMensajeRecibido[2],
st.byMensajeRecibido[3], st.byMensajeRecibido[4], st.byMensajeRecibido[5]);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se puede conectar con el Cliente de estados: " +
ex.Message, "Advertencia", MessageBoxButtons.OK);
            }
        }

        public void CerrarServidor()
        {//Cierra el servidor
            bFinalizarHilo = true;
            clienteConectado.Set();
        }
        #endregion
    }
}
```

2.3 Código Simulador Turbina

2.3.1 Program.cs

```
/******************
 ** Program.cs **
    31/05/2016
 ******************/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Turbina
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormTurbina());
        }
    }
}
```

2.3.2 FormTurbina.cs

```
/***********************
 ** FormTurbina.cs **
    31/05/2016
 ***********************/

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows.Forms;

using System.Threading;

namespace Turbina
{
    public partial class FormTurbina : Form
    {
        public TratPuertoSerie puertoSerie = new TratPuertoSerie();
        public delegate void SetText(string sText);

        public byte[] byTrama;

        private Thread hiloCadenas;

        public FormTurbina()
        {
            InitializeComponent();
        }

        #region Eventos del formulario

        private void Turbina_Load(object sender, EventArgs e)
        {
            hiloCadenas = new Thread(this.GeneraCadenas);
            hiloCadenas.Start();
        }

        private void Turbina_FormClosing(object sender, FormClosingEventArgs e)
        {
            try
            {
                puertoSerie.CerrarPuerto();

            }
            catch { }
            hiloCadenas.Abort();
        }

        #endregion

        #region Gestión envío nuevas cadenas

        private void GeneraCadenas()
        {
            while (true)
            {
                byTrama = NuevaCadena();

                puertoSerie.EscribirPuerto(byTrama);

                this.MuestraDato(
                                        String.Format("{0} {1} {2} {3} {4} {5} {6} {7} {8} {9}
{10} {11} {12} {13} {14} {15} {16} {17} {18} {19} {20} {21} {22} {23} {24} {25} {26} {27}
{28} {29} {30} {31} {32} {33}",
```

```
                                                    DateTime.Now.ToString("HH:mm:ss.fff"),
byTrama[0], byTrama[1], byTrama[2], byTrama[3], byTrama[4], byTrama[5],
                                                    byTrama[6], byTrama[7], byTrama[8],
byTrama[9], byTrama[10], byTrama[11], byTrama[12], byTrama[13], byTrama[14], byTrama[15],
                                                    byTrama[16], byTrama[17], byTrama[18],
byTrama[19], byTrama[20], byTrama[21], byTrama[22], byTrama[23], byTrama[24],
                                                    byTrama[25], byTrama[26], byTrama[27],
byTrama[28], byTrama[29], byTrama[30], byTrama[31], byTrama[32]
                                                )
                );

                Thread.Sleep(Properties.Settings.Default.milisecEmisionDatos); // Tiempo
espera para la siguiente emisión de datos
            }
        }

        private byte[] NuevaCadena()
        {
            byte[] byTrama = new byte[33];

            byTrama[0] = (byte)0x10;
            byTrama[1] = (byte)0x02;

            byte[] byVelocidad = BitConverter.GetBytes((Int16)spEditVelocidad.Value);
            byTrama[2] = byVelocidad[1];
            byTrama[3] = byVelocidad[0];

            byte[] byPresion = BitConverter.GetBytes((Int16)spEditPresion.Value);
            byTrama[4] = byPresion[1];
            byTrama[5] = byPresion[0];

            byte[] byPosCilindro = BitConverter.GetBytes((Int16)spEditPosCilindro.Value);
            byTrama[6] = byPosCilindro[1];
            byTrama[7] = byPosCilindro[0];

            byte[] byPotencia = BitConverter.GetBytes((Int16)spEditPotencia.Value);
            byTrama[8] = byPotencia[1];
            byTrama[9] = byPotencia[0];

            byte[] byVibraciones = BitConverter.GetBytes((Int16)spEditVibraciones.Value);
            byTrama[10] = byVibraciones[1];
            byTrama[11] = byVibraciones[0];

            byte[] byCorrienteIU = BitConverter.GetBytes((Int16)spEditCorrIU.Value);
            byTrama[12] = byCorrienteIU[1];
            byTrama[13] = byCorrienteIU[0];

            byte[] byCorrienteIV = BitConverter.GetBytes((Int16)spEditCorrIV.Value);
            byTrama[14] = byCorrienteIV[1];
            byTrama[15] = byCorrienteIV[0];

            byte[] byTensionVDC = BitConverter.GetBytes((Int16)spEditTension.Value);
            byTrama[16] = byTensionVDC[1];
            byTrama[17] = byTensionVDC[0];

            byte[] byTemp1 = BitConverter.GetBytes((Int16)spEditTemp1.Value);
```

```
        byTrama[18] = byTemp1[1];
        byTrama[19] = byTemp1[0];

        byte[] byTemp2 = BitConverter.GetBytes((Int16)spEditTemp2.Value);
        byTrama[20] = byTemp2[1];
        byTrama[21] = byTemp2[0];

        byte[] byComPosValvula = BitConverter.GetBytes((Int16)spEditComValvula.Value);
        byTrama[22] = byComPosValvula[1];
        byTrama[23] = byComPosValvula[0];

        byte[] byRefGiroMax = BitConverter.GetBytes((Int16)spEditRefVelocidad.Value);
        byTrama[24] = byRefGiroMax[1];
        byTrama[25] = byRefGiroMax[0];

        int error1 = 0;
        foreach (int indiceChecked in chckListError1.CheckedIndices)
        {
            error1 = error1 | (int)Math.Pow(2, indiceChecked);
        }
        byTrama[26] = (byte)error1;
        int error2 = 0;
        foreach (int indiceChecked in chckListError2.CheckedIndices)
        {
            error2 = error2 | (int)Math.Pow(2, indiceChecked);
        }
        byTrama[27] = (byte)error2;

        int error3 = 0;
        foreach (int indiceChecked in chckListError3.CheckedIndices)
        {
            error3 = error3 | (int)Math.Pow(2, indiceChecked);
        }
        byTrama[28] = (byte)error3;

        int estado = 0;
        foreach (int indiceChecked in chckListEstado.CheckedIndices)
        {
            estado = estado | (int)Math.Pow(2, indiceChecked);
        }
        byTrama[29] = (byte)estado;

        byTrama[30] = (byte)0;
        byTrama[31] = (byte)0x10;
        byTrama[32] = (byte)0x03;

        return byTrama;
    }

    #endregion

    private void MuestraDato(string sDato)
    {
        if (lstBoxTramas.InvokeRequired)
        {
            Invoke(new SetText(this.MuestraDato), new object[] { sDato });
```

```
            }
            else
            {
                lstBoxTramas.Items.Insert(0, sDato);
            }
        }

        #region Gestión cadenas recibidas

        private void InterpretaEstado(byte nuevoEstado)
        {
            System.Collections.BitArray bitArrEstado = new System.Collections.BitArray(new
byte[] { nuevoEstado });

            if (bitArrEstado[0] == true)
            {
                chckListEstado.SetItemCheckState(0, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(0, CheckState.Unchecked);
            }

            if (bitArrEstado[1] == true)
            {
                chckListEstado.SetItemCheckState(1, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(1, CheckState.Unchecked);
            }

            if (bitArrEstado[2] == true)
            {
                chckListEstado.SetItemCheckState(2, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(2, CheckState.Unchecked);
            }

            if (bitArrEstado[3] == true)
            {
                chckListEstado.SetItemCheckState(3, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(3, CheckState.Unchecked);
            }

            if (bitArrEstado[4] == true)
            {
                chckListEstado.SetItemCheckState(4, CheckState.Checked);
            }
            else
            {
```

```
                chckListEstado.SetItemCheckState(4, CheckState.Unchecked);
            }

            if (bitArrEstado[5] == true)
            {
                chckListEstado.SetItemCheckState(5, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(5, CheckState.Unchecked);
            }

            if (bitArrEstado[6] == true)
            {
                chckListEstado.SetItemCheckState(6, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(6, CheckState.Unchecked);
            }

            if (bitArrEstado[7] == true)
            {
                chckListEstado.SetItemCheckState(7, CheckState.Checked);
            }
            else
            {
                chckListEstado.SetItemCheckState(7, CheckState.Unchecked);
            }
        }

        private void ProcesaTrama(byte[] trama)
        {
            bool correcto = true;
            int estado = 0;
            if ((trama[0] == 0x10) && (trama[1] == 0x02) && (trama[5] == 0x10) && (trama[6]
== 0x03))
            {
                switch (trama[2])
                {
                    case 0x00: //Comando funcionamiento normal
                        estado = estado & 251; //pone a cero el bit 2 (máscara 1111 1011)
                        this.InterpretaEstado((byte)estado);
                        this.spEditVelocidad.Value = this.spEditRefVelocidad.Value;
                        break;

                    case 0xFF: //Comando parar turbina
                        estado = estado | 4; //pone a uno el bit 2 (máscara 0000 0100)
                        estado = estado & 191; //pone a cero el bit 7 (máscara 1011 1111)
                        this.InterpretaEstado((byte)estado);
                        this.spEditVelocidad.Value = 0;
                        break;

                    case 0xAA: //Comando arrancar turbina
                        estado = estado & 251; //pone a cero el bit 2 (máscara 1111 1011)
                        estado = estado | 64; //pone a uno el bit 7 (máscara 0100 0000)
```

```
                    this.InterpretaEstado((byte)estado);
                    this.spEditVelocidad.Value = this.spEditRefVelocidad.Value/2;
                    break;

                case 0xF0: //Comando abrir válvula
                    estado = estado | 2; //pone a uno el bit uno (máscara 0000 0010)
                    this.InterpretaEstado((byte)estado);
                    this.spEditComValvula.Value = 50;
                    this.spEditPosCilindro.Value = 50;
                    break;

                case 0x0F: //Comando cerrar válvula
                    estado = estado & 253; //pone a cero el bit uno (máscara 1111 1101)
                    this.InterpretaEstado((byte)estado);
                    this.spEditComValvula.Value = 0;
                    this.spEditPosCilindro.Value = 0;
                    break;

                case 0x0A: //Comando referencia velocidad giro máxima
                    this.spEditRefVelocidad.Value = BitConverter.ToInt16(trama, 3);
                    if (this.spEditVelocidad.Value >= this.spEditRefVelocidad.Value)
                    {
                        this.spEditVelocidad.Value = this.spEditRefVelocidad.Value;
                    }
                    break;

                case 0xA0: //Comando referencia posición valvula
                    this.spEditComValvula.Value = BitConverter.ToInt16(trama, 3);
                    this.spEditPosCilindro.Value = BitConverter.ToInt16(trama, 3);
                    break;

                default:
                    correcto = false;
                    break;
            }
            if (correcto == false)
            {
                this.MuestraDato("Comando recibido incorrecto");
            }
        }
    }

    #endregion


}
}
```

2.3.3 TratPuertoSerie.cs

```
/*************************
 ** TratPuertoSerie.cs **
```

```
    31/05/2016
 *************************/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO.Ports;
using System.Windows.Forms;

namespace Turbina
{
    public class TratPuertoSerie
    {
        private SerialPort puertoSerie;
        public delegate void nuevaTramaRecibidaEventHandler(byte[] byNuevaTrama);
        public event nuevaTramaRecibidaEventHandler eventoNuevaTramaRecibida;

        public TratPuertoSerie()
        {
            puertoSerie = new SerialPort(Properties.Settings.Default.sPuerto);
            puertoSerie.BaudRate = Properties.Settings.Default.iBaudRate;
            puertoSerie.Parity = Parity.None;
            puertoSerie.StopBits = StopBits.One;
            puertoSerie.DataBits = Properties.Settings.Default.iDataBits;
            puertoSerie.Handshake = Handshake.None;
            try
            {
                puertoSerie.Open();
                puertoSerie.DiscardOutBuffer();
                puertoSerie.DiscardInBuffer();
                puertoSerie.DataReceived+=new
SerialDataReceivedEventHandler(LeeNuevosDatos);
            }
            catch(Exception ex)
            {
                MessageBox.Show("No se puede abrir el puerto serie: " + ex.Message, "Error
puerto serie", MessageBoxButtons.OK);
            }
        }

        public void CerrarPuerto()
        {
            puertoSerie.Close();
        }

        #region Lectura puerto serie
        private void LeeNuevosDatos(object sender, SerialDataReceivedEventArgs e)
        {
            int iBytesLeidos = puertoSerie.BytesToRead; //Bytes leídos
            byte[] byTramaRecibida=new byte[7];

            if (iBytesLeidos > 0)
            {
                try
```

```
                {
                    puertoSerie.Read(byTramaRecibida, 0, iBytesLeidos);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("No se puede leer el puerto serie: " + ex.Message,
"Error puerto serie", MessageBoxButtons.OK);
                }
                if (this.eventoNuevaTramaRecibida != null)
                {
                    this.eventoNuevaTramaRecibida(byTramaRecibida);
                }
            }
        }

        #endregion

        #region Escritura puerto serie

        public void EscribirPuerto(byte[] trama)
        {
            try
            {
                puertoSerie.Write(trama, 0, 33);
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se puede leer el puerto serie: " + ex.Message, "Error
puerto serie", MessageBoxButtons.OK);
            }
        }
        #endregion
    }

       }
```