UNIVERSIDAD DEL BÍO-BÍO
La Universidad de la Región del Biobío

Laboratorio de MAC-IPP
Profesor: Luís Vera Quiroga
Alumno ayudante: Camilo Esteban Zapata O.
Corregido por (alumno ayudante): José I. Veloso Inzunza

Practico Nº2 Comunicación Serial y Socket

Ejercicio 1: "Trabajando con SerialPort en Scorbot"

Descripción: El objetivo de este práctico es enviar datos a través del puerto COM a un brazo robótico que espera datos para realizar las instrucciones de abrir y cerrar el gripper, o ejecutar algún programa previamente grabado.

Abrir el programa Visual Studio y seleccionar **Crear un proyecto**, luego se debe seleccionar la opción **Aplicación de Windows Forms (.Net Framework),** implementado en C#. Asigne el nombre que usted desee y luego diseñe el formulario agregando los siguientes componentes:

Nombre	**Propiedades**
Form1	Text: Comunicación Serial Size 485; 354
Button1	Text: Conectar
Button2	Text: Desconectar
Button3	Text: Enviar
Button4	Text: Guardar
Button5	Text: Run pcplc
Button6	Text: Abortar
Button7	Text: Run ttsib
Button8	Text: coff
Button9	Text: move 0
Button10	Text: open
Button11	Text: close
ComboBox1	Ítems: COM1, COM2…, DropDownStyle: DropDownList
textBox1	Text: Desconectado TextAlign: Center
RichTextBox1	Text:
RichTextBox2	Text:
SerialPort1	
Timer1	
SaveFileDialog1	
Label1	Text: Datos Enviados
Label2	Text: Datos Recibidos

El diseño del formulario debería quedar de la siguiente manera:

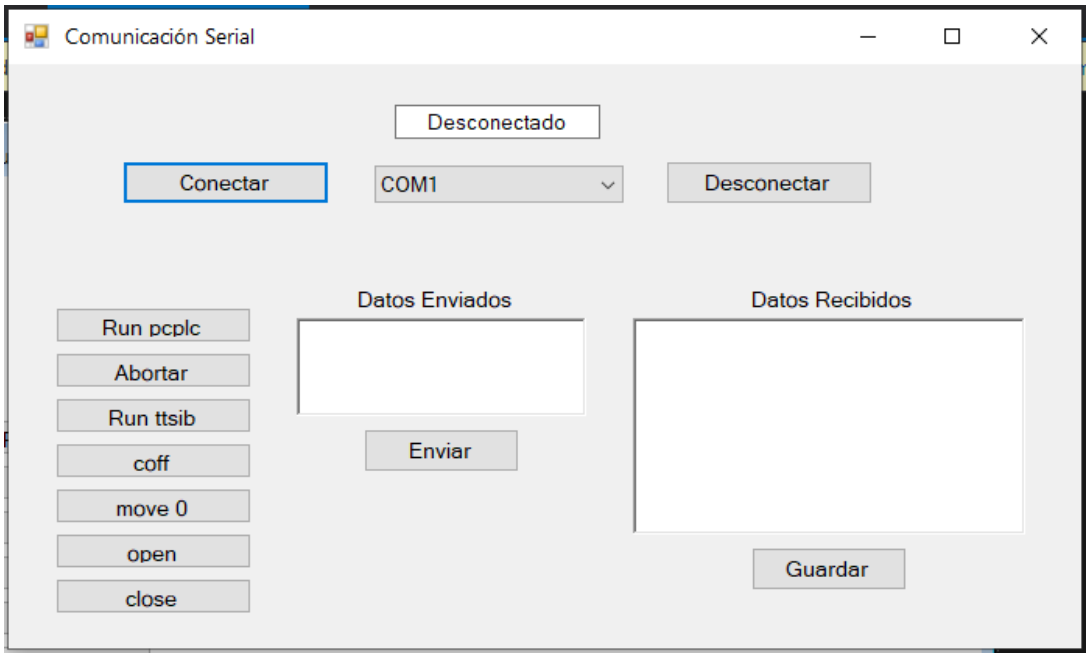

Figura 1. Diseño del Form

Importante: para llevar a cabo la actividad agregue bibliotecas adicionales de entrada y salida para comunicación, para ello haga doble clic en el Form1 y en el encabezado añada lo siguiente:

```
using System.IO.Ports;
using System.IO;
```

1. Llene el *ComboBox1* que contendrá los puertos COM a disposición por medio de la propiedad Ítems, además para bloquear la edición del *ComboBox1*, debe ir a la propiedad *DropDownStyle* y cambiarla a *DropDownList*.

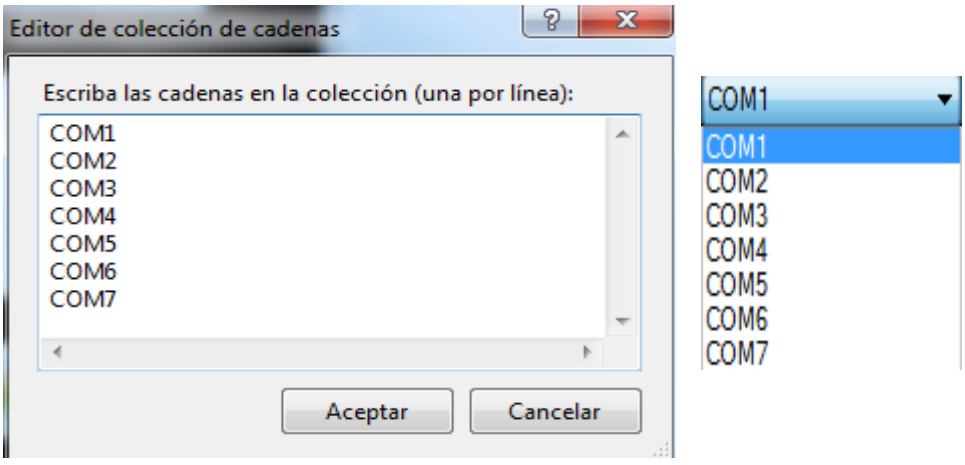

Figura 2. ComboBox

Lo anterior también se puede modificar agregando las líneas de código que se muestran a continuación:

```
private void Form1_Load(object sender, EventArgs e)
{
    comboBox1.DropDownStyle = ComboBoxStyle.DropDownList;
    comboBox1.SelectedIndex = 0;


}
```

2. El código de los botones 5 al 11 puede modificarse para escribir automáticamente en el RichTextBox1 comandos usados con frecuencia, de la siguiente manera:

```
private void button5_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "Run pcplc";
}

1 referencia
private void button6_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "a";
}

1 referencia
private void button7_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "Run ttsib";
}

1 referencia
private void button8_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "coff";
}

1 referencia
private void button9_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "move 0";
}

1 referencia
private void button10_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "open";
}

1 referencia
private void button11_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "close";
}
```

3. Agregue el código del botón Conectar, haciendo doble clic sobre ese botón:

```
1 reference
private void Button1_Click(object sender, EventArgs e)
{

    if (serialPort1.IsOpen == false)
    {
        serialPort1.BaudRate = 9600;
        serialPort1.DataBits = 8;
        serialPort1.Parity = Parity.None;
        serialPort1.StopBits = StopBits.One;
        serialPort1.PortName = comboBox1.SelectedItem.ToString();
        serialPort1.Open();
        textBox1.Text = " Conectado";


        MessageBox.Show("Puerto conectado ");
        textBox1.BackColor = Color.Lime;
        timer1.Enabled = true;
    }
}
```

En la condición *if* se configuran los parámetros del puerto COM. Una vez que está configurado se abre el puerto, se muestra un mensaje de apertura y habilita un *timer*.

4. Luego agregue el código del botón desconectar:

```
1 reference
private void Button2_Click(object sender, EventArgs e)
{
     if( serialPort1.IsOpen == true)
    {
        serialPort1.Close();
        timer1.Enabled = false;
        textBox1.Text = " Desconectado";

        MessageBox.Show("Puerto desconectado");
        textBox1.BackColor = Color.Red;

    }
}
```

5. Agregue el siguiente código al botón Enviar:

```
1 reference
private void Button3_Click(object sender, EventArgs e)
{
    serialPort1.Write(richTextBox1.Text);
    serialPort1.Write("\r");
}
```

Las operaciones que se van realizando son almacenadas en el richtextBox1 para tener un seguimiento de lo realizado.

6. Para finalizar se emplea la función *Timer1_Tick* y un botón para guardar el contenido de la comunicación:

```
1 reference
private void Timer1_Tick(object sender, EventArgs e)
{
    mensaje = serialPort1.ReadExisting();

    if( mensaje.Length > 0)
    {
        richTextBox2.Text += mensaje + "\n";
    }
}
```

```
1 reference
private void Button4_Click(object sender, EventArgs e)
{
    if( saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        File.WriteAllLines(saveFileDialog1.FileName + " .txt", richTextBox2.Lines);
    }


}
```

Mediante el *timer1* se almacena en el *richtextBox2* el contenido recibido de la comunicación serial, así se tiene un registro de lo que se recibe en la comunicación.

7. Declare las siguientes variables globales y compile.

```
string mensaje;
string output;
string msj;
int i;
```

8. Configure como enviar la secuencia de instrucciones en el botón Enviar:

```
try
{
    msj = richTextBox1.Text.ToString();

    List<string> lista = new List<string>(msj.Split(new char[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries));

    for (i = 0; i < lista.Count; i++)
    {
        serialPort1.Write(lista[i]);
        serialPort1.Write("\r");
        Thread.Sleep(5000);
    }
}
catch (Exception ex)
{
    output = "Error: " + ex.ToString();
    MessageBox.Show(output);
}
```

La variable de tipo string "*msj*" recibe todo lo captado desde el *Richtexbox1*. Se crea una lista *List<String> lista* cuyo método *Split* divide el *string* en *substrings* cuando cumple la condición *{'\n', '\r'}*, es decir, almacena y "limpia" los string recibidos, separándolos en función de los saltos o retorno de línea.

Se crea un ciclo for para:
- Recorrer las listas mediante la variable "i".
- En cada ciclo se envía la lista al serialport.
- Con *Thread.Sleep(5000),* cada 5 segundos pasa al siguiente ciclo (instrucción).

Se debe incluir la siguiente biblioteca:

```
using System.Threading;
```

En la función *Timer1_Tick* añadimos una sentencia *if(mensaje.Contains("Done.").* Existe un problema con la recepción del símbolo "?". El mensaje recibido es guardado en la variable *mensaje*, pero la frase "?Done" no es reconocida por el programa. Por ello se emplea la función *Contains* que ayuda a identificar el mensaje recibido para mostrarlo correctamente.

El comando enviado al puerto serial se recibe de vuelta, como no se desea recibir los mensajes enviados deben ser removidos. El comando enviado está almacenado en la variable "msj" y la función *Remove* la elimina del listado de mensajes recibidos. Posteriormente estos mensajes son enviados al *richTextBox* para ser mostrados por la pantalla del Form.

Finalmente, los mensajes recibidos son mostrados tal y como serían mostrados en una consola (línea por línea) en el *richTextBox2*:

```
mensaje = serialPort1.ReadExisting().ToString();
mensaje = mensaje.Trim();

if (mensaje.Length > 0)
{
    if (mensaje.Contains("Done.")) //el programa no reconoce el simbolo "?" en la frase "?Done."
    {
        //Console.WriteLine("Flag DONE \n");
        richTextBox2.Text += "?Done.\n";
    }

    List<string> list = new List<string>(mensaje.Split(new char[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries));


    list.Remove(msj); //se elimina de la lista el comando eviad,
                      //ya que éste se recibe de vuelta junto con las respuestas del robot
    for (i = 0; i < list.Count; i++)
    {
        richTextBox2.Text += (list[i]) + "\r";
    }
}
```