

## Comunicaciones RS-232 (Puerto Serie)

El puerto serial, como su nombre lo indica envía su información de manera serial, es decir, como un tren de pulsos, utilizando el protocolo RS-232. Para la transmisión de información solo son necesarios 3 pines, uno a través del cual se envía la información, otro a través del cual se recibe y otro como referencia de voltaje o tierra. Pero el puerto serial posee 9 pines, los 5 restantes son para el control de datos, petición de información, libre para enviar, etc.

Una característica de este puerto que NO se debe olvidar son los valores de voltaje que utiliza para sus niveles lógicos. Un 0 (cero) lógico corresponde a un voltaje de entre +3 y +25 volts, mientras que un 1 (uno) lógico va de -3 a -25 volts.

Esto es importante ya que si se desea interfacear el puerto con algún circuito TTL o CMOS se debe adaptar el valor del voltaje.

### Características mecánicas

En el estandar no se hace referencia al tipo de conector que debe usarse. Sin embargo los conectores mas comunes son el DB-25 (25 pines) y el DB-9 (9 pines). El conector hembra debe estar asociado con el DCE y el macho con el DTE.



Por último para el envío de información es necesario que tanto el emisor como el receptor estén configurados para trabajar a la misma tasa de transferencia, ya que la comunicación es asíncrona y la señal de reloj no es enviada con la información.

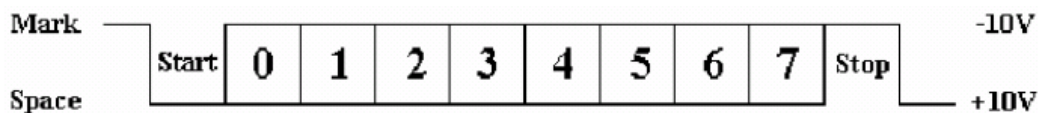
Para lograr la sincronización el puerto utiliza un protocolo el cual envía un bit de partida, el cual no es configurable. Lo que sí es configurable es el bit de parada, la paridad y el número de bits. La configuración más utilizada es la 8N1 (8 bits de información, sin paridad y 1 bit de parada).

En las comunicaciones asíncronas el estado de reposo (cuando no se transmite nada) se identifica con un "1" (**marca**). Cuando se recibe un **bit de inicio**, que es un "0" (**espacio**), el receptor toma nota que va a comenzar a recibir un dato.

Los parámetros que caracterizan estas comunicaciones son: **Velocidad**; **paridad**; **bits de datos** y **bits de parada**. En la literatura sobre el tema es frecuente expresar estos datos en forma resumida. Por ejemplo: **1200 8 N 1** para indicar una transmisión de 1200 baudios con 8 bits de datos sin paridad y un bit de Stop.

**Velocidad de transmisión** ("Connection speed") es la cantidad de datos transmitidos en unidad de tiempo. Se expresa en bits por segundo (**bps**). En las transmisiones serie a través de líneas telefónicas, en las que se emplean módems era frecuente utilizar como medida de velocidad el **Baudio** ("Baud rate"). Baudio se define como el número de veces que cambia la portadora en un segundo. La velocidad que puede emplearse depende en gran medida de la calidad del medio de transmisión (calidad de la línea), que si (como es frecuente) se trata de líneas telefónicas, depende a su vez de la distancia.

Los primeros dispositivos serie operaban a velocidades muy bajas, del orden de 110 a 1200 baudios. Las comunicaciones telefónicas serie actuales están muy cerca del máximo teórico que pueden soportar los pares de cobre utilizados en la telefonía estándar.



D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

**Bits de datos** ("Char length"). Se han utilizado caracteres de 5, 6, 7 y 8 bits, aunque actualmente los datos son enviados como caracteres ASCII, por lo que pueden utilizarse 7 u 8 bits según se trate del juego de caracteres US-ASCII o el extendido. El conjunto de bits que componen un carácter se denominan **bits de dato** ("Data bits")

**Paridad** ("Parity"). Para poder comprobar la calidad de la transmisión se suele utilizar un sistema de control de paridad que añade un bit a los bits de datos. Los sistemas utilizados son:

- Paridad par ("**E**ven")
- Paridad impar ("**O**dd")
- Paridad marca ("**M**arck")
- Paridad espacio ("**S**pace")
- Sin paridad (no se añade ningún bit de paridad al datagrama)

**Bits de parada** ("Stop bits"). Después que se envía un carácter se envía un bits de parada, que tienen el valor "1" (marca); la duración de este bit puede ser 1, 1.5 o 2 periodos.

## Programación del Puerto Serie en C#

Para poder configurar el puerto necesitamos la librería IO.Ports del Framework, para ello utilizamos la directiva using:

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

La librería IO.Ports tiene una clase SerialPort que nos va a permitir configurar el puerto. Para ello declaramos un objeto de la clase SerialPort:

```
SerialPort puertoSerie = new SerialPort();
```

Adicionalmente podemos inicializar el objeto con los siguientes parámetros:

```
SerialPort puerto = new SerialPort(nombre_puerto, velocidad, paridad, numero_bits, bits_parada);
```

**Nombre del puerto (string)** Indica el nombre del puerto serie usado. Cambiando esa propiedad podemos cambiar el puerto de comunicación que vamos a usar (Un PC tiene normalmente 2 puertos serie : El "COM1" y el "COM2". Puede tener sin grandes problemas Hardware hasta 4 ("COM3" y "COM4") Si le damos a ese valor un número de puerto inexistente, dará error.

**Velocidad de Transmisión (int):** Indica la velocidad en baudios con la que vamos a transmitir los datos. Los valores posibles para **velocidad** son

50 100 110 300 600 1200 2400 4800 9600 14400 19200 y 28800

### Bits de Paridad(Parity)

- Paridad par ("**Parity.Even**")
- Paridad impar ("**Parity.Odd**")
- Paridad marca ("**Parity.Marck**")
- Paridad espacio ("**Parity.Space**")
- Sin paridad, no se añade ningún bit de paridad al datagrama ("**Parity.None**");

**Número de bits (int).** Los valores para el parámetro **Bits de Información** pueden ser:

7 - Se envían / reciben 7 bits por trama de información.

8 - Se envían / reciben 8 bits por trama de información

**Bits de Parada (StopBits)** Los valores para el parámetro **Bits de parada** pueden ser :

1 - Se envía un bit de parada (**StopBits.One**)

2 - Se envían 2 bits de parada (**StopBits.Two**)

## Principales métodos y atributos de la clase SerialPort

**Open.** Abrirá el puerto con la configuración determinada en el constructor del objeto.

Ejemplo:

```
Puerto.Open();
```

**Close.** Cerrará el puerto que se haya abierto mediante el método Open.

Ejemplo:

```
Puerto.Close();
```

**IsOpen.** Retorna un valor booleano trae si el puerto está abierto.

Ejemplo:

```
If(puerto.IsOpen==true)  
//Esta abierto
```

**GetPortNames.** Retorna un arreglo con los puertos configurados en la PC.

Ejemplo:

```
SerialPort.GetPortNames();
```

**Write.** Envía la cadena especificada al puerto.

```
Puerto.Write("Mensaje");
```

```
char[] c=new char[5];  
c[0]='A';  
c[1]='B';  
c[2]='C';  
c[3]='D';  
c[4]='E';  
Puerto.Write(c,0,4);
```

```
byte[] b=new byte[5];  
c[0]=1;  
c[1]=2;  
c[2]=3;  
c[3]=4;  
c[4]=5;  
Puerto.Write(c,0,4);
```

**ReadExisting.** Obtiene todos los bytes disponibles en el búfer del puerto.

Ejemplo:

```
string cadena_leida;  
cadena_leida=Puerto.ReadExis();
```

```
byte b;  
b=Puerto.ReadByte();
```

```
char c;  
c= Puerto.ReadChar();
```

**Evento para recepción de datos**

```
Puerto.DataReceived += new SerialDataReceivedEventHandler(Puerto_DataReceived);
```

```
private void Puerto_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    //Metodo que estará leyendo todo lo que llegue al puerto
    //Con ReadExisting() Devuelve automaticamente todo lo que le llega al puerto
    txtRecepcion.Text = Puerto.ReadExisting();
}
```

**Control Timer**

El timer es un control que nos va a permitir controlar un evento de manera periódica. Sus principales métodos y atributos son;

**Name.** Nombre del control.

**Interval.** El intervalo que pasará entre una ejecución y otra del evento, está dada en milisegundos. Por ejemplo si queremos que un evento se repita cada segundo deberemos configurar el intervalo a 1000.

**Start().** Inicia el timer.

**Stop().** Detiene el timer.

**Ejemplos de Configuración**

```
*****
```

```
// This is a new namespace in .NET 2.0
// that contains the SerialPort class
using System.IO.Ports;

private static void SendSampleData()
{
    // Instantiate the communications
    // port with some basic settings
    SerialPort port = new SerialPort(
        "COM1", 9600, Parity.None, 8, StopBits.One);

    port.DataReceived += new SerialDataReceivedEventHandler(port_DataReceived);

    // Open the port for communications
    port.Open();

    // Write a string
    port.Write("Hello World");

    // Write a set of bytes
    port.Write(new byte[] {0x0A, 0xE2, 0xFF}, 0, 3);

    // Close the port
    port.Close();
}
```

```
private void port_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    string cad;
    //Metodo que estará leyendo todo lo que llegue al puerto
    //Con ReadExisting() Devuelve automaticamente todo lo que le llega al puerto
    cad = port.ReadExisting();
}
```

\*\*\*\*\*

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

namespace SerialPortExample
{
    class SerialPortExample
    {
        public static void Main()
        {
            // Get a list of serial port names.
            string[] ports = SerialPort.GetPortNames();

            Console.WriteLine("The following serial ports were found:");

            // Display each port name to the console.
            foreach(string port in ports)
            {
                Console.WriteLine(port);
            }

            Console.ReadLine();
        }
    }
}
```

Ejercicios de la Práctica:

Controles nuevos que veremos hoy

### Combo Box

El control Combo Box le permite desplegar una lista de datos en un menú desplegable.

#### Principales propiedades:

**Name.** Permite asignarle un nombre al control. Se recomienda anteponer un **cmb**Nombre.

**Ítems (Collection).** Permite generar la lista de datos que se desplegará en el ComboBox.

**Text.** Muestra un texto inicial en el objeto.

Principales Métodos:

**Text.** Devuelve el texto que se muestra actualmente en el Combo Box. Ejemplo:

**Variable = ComboBox1.Text**

### RadioButton

El Control RadioButton le permite seleccionar una sola opción de una lista de posibles opciones.

#### Principales propiedades:

**Name.** Permite asignarle un nombre al control. Se recomienda anteponer un **rbtn**Nombre.

**Text.** Devuelve el texto que se muestra Como opción.

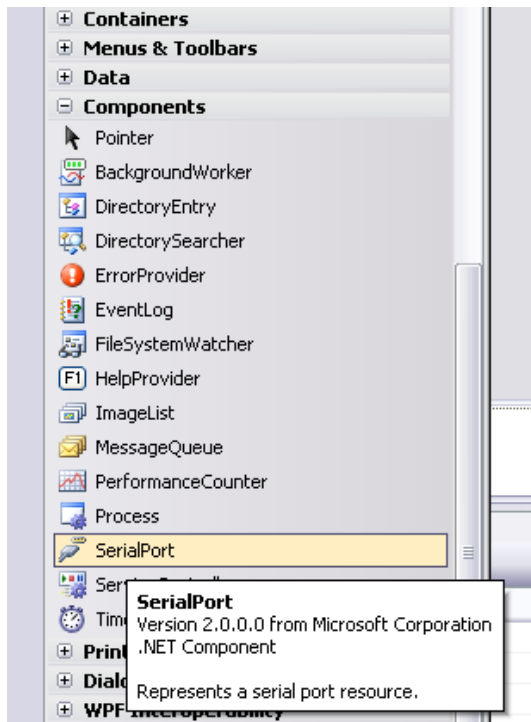
**Checked.** Determina la opción que estará activa o ha sido seleccionada. Ejemplo:

```
rbtnNombre.Checked=True;
```

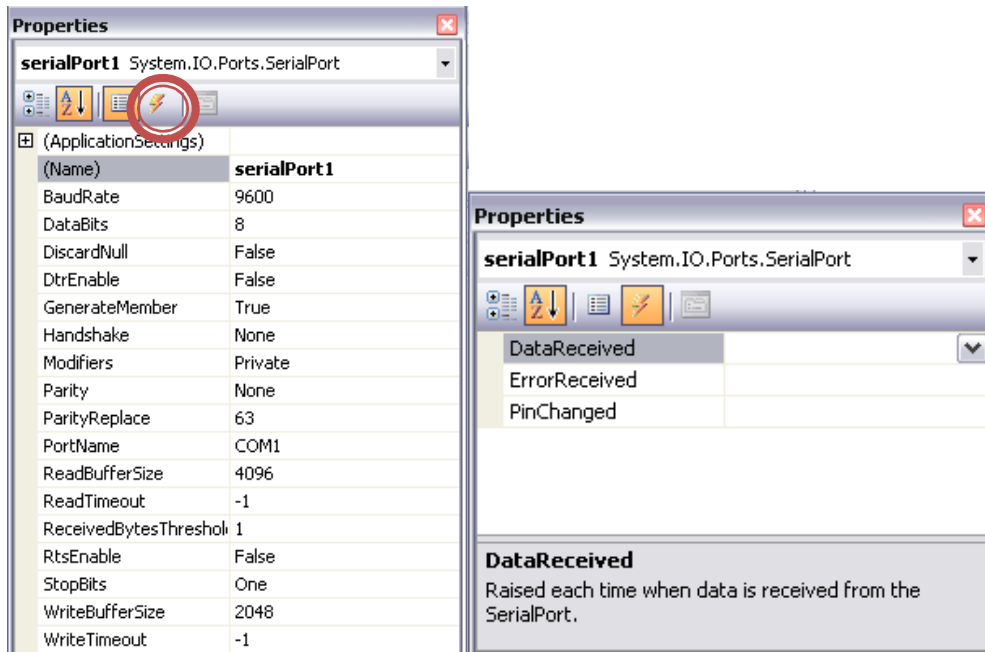
```
o
```

```
if(rbtnNombre==True)  
    //Codigo ha ejecutar
```

1. Crea una nueva solución y llámala Practica25\_Puerto\_serie. Posteriormente de la caja de herramientas selecciona el botón **Serial Port** y arrástralo a tu formulario.



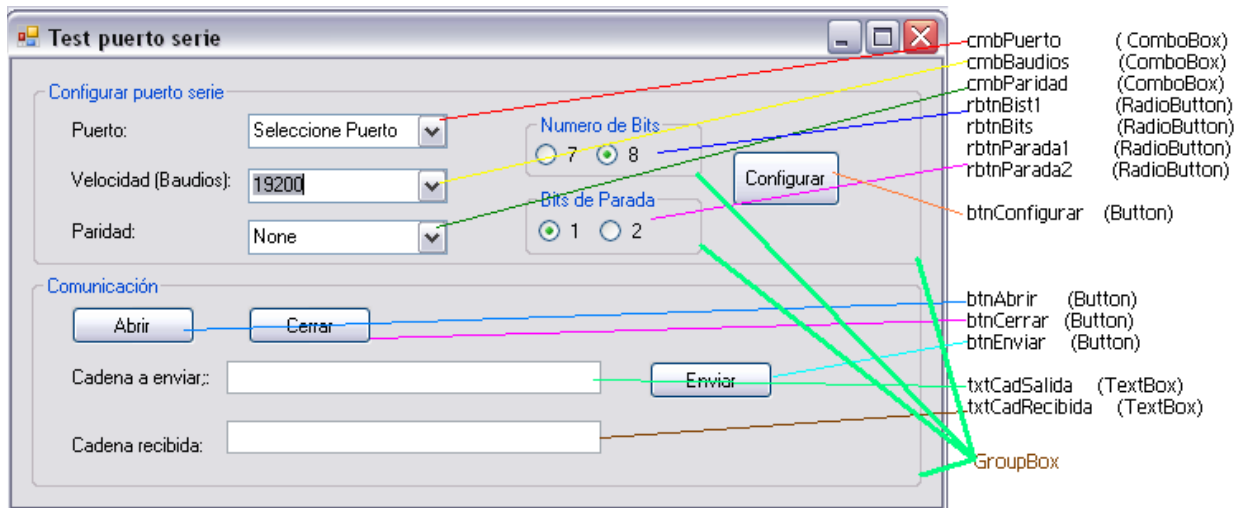
2. Posteriormente en la ventana de propiedades vamos a agregar el método DataReceived:



```
private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e) { }
```

3. Ahora vamos a diseñar el formulario para que quede de la siguiente manera:

**CICLO ESCOLAR 2010-2**  
**Practica 25. Comunicación RS-232**



4. Ahora vamos a editar el código del formulario para que quede de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace puerto_serie
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            txtCadRecibida.Text = serialPort1.ReadExisting();
        }

        private void btnEnviar_Click(object sender, EventArgs e)
        {
            if ((serialPort1 != null) && (serialPort1.IsOpen == true))
                serialPort1.Write(txtCadSalida.Text);
            //Tambien podriamos enviarle de esta manera los datos
            //serialPort1.Write(new byte[] {0x0A, 0xE2, 0xFF}, 0, 3);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            busca_puertos();

            //Configuramos el puerto por default
        }
    }
}
```

```

configura_puerto("COM1", 19200, Parity.None, 8, StopBits.One);

//Inicializamos velocidad
cmbBaudios.Items.Add("1200");
cmbBaudios.Items.Add("2400");
cmbBaudios.Items.Add("4800");
cmbBaudios.Items.Add("9200");
cmbBaudios.Items.Add("19200");
cmbBaudios.Text = "19200";

//Inicializamos la paridad
cmbParidad.Items.Add(Parity.Even.ToString());
cmbParidad.Items.Add(Parity.Mark.ToString());
cmbParidad.Items.Add(Parity.None.ToString());
cmbParidad.Items.Add(Parity.Odd.ToString());
cmbParidad.Items.Add(Parity.Space.ToString());
cmbParidad.Text = "None";
}

private void configura_puerto(string npuerto, int baudios, Parity paridad, int nBits, StopBits bits_parada)
{
    serialPort1 = new SerialPort(npuerto, baudios, paridad, nBits, bits_parada);
}

private void busca_puertos()
{
    // Obtenemos los puertos que se encuentran configurados en el equipo
    string[] ports = SerialPort.GetPortNames();

    cmbPuerto.Items.Clear();
    // Los desplegamos en el ComboBox
    foreach (string port in ports)
    {
        cmbPuerto.Items.Add(port);
    }
    cmbPuerto.Text = "Seleccione Puerto";
}

private void btnConfigurar_Click(object sender, EventArgs e)
{
    Parity Paridad=Parity.None;
    StopBits BitsParada=StopBits.One;
    int numBits=8;

    if (cmbParidad.Text == "Even")
        Paridad = Parity.Even;
    else if (cmbParidad.Text == "Mark")
        Paridad = Parity.Mark;
    else if (cmbParidad.Text == "Odd")
        Paridad = Parity.Odd;
    else if (cmbParidad.Text == "Space")
        Paridad = Parity.Space;
    else
        Paridad = Parity.None;

    if (rbtnParada1.Checked)
        BitsParada = StopBits.One;
    else
        BitsParada = StopBits.Two;

    if (rbtnBits1.Checked)
        numBits=7;
}

```

```

else
    numBits=8;

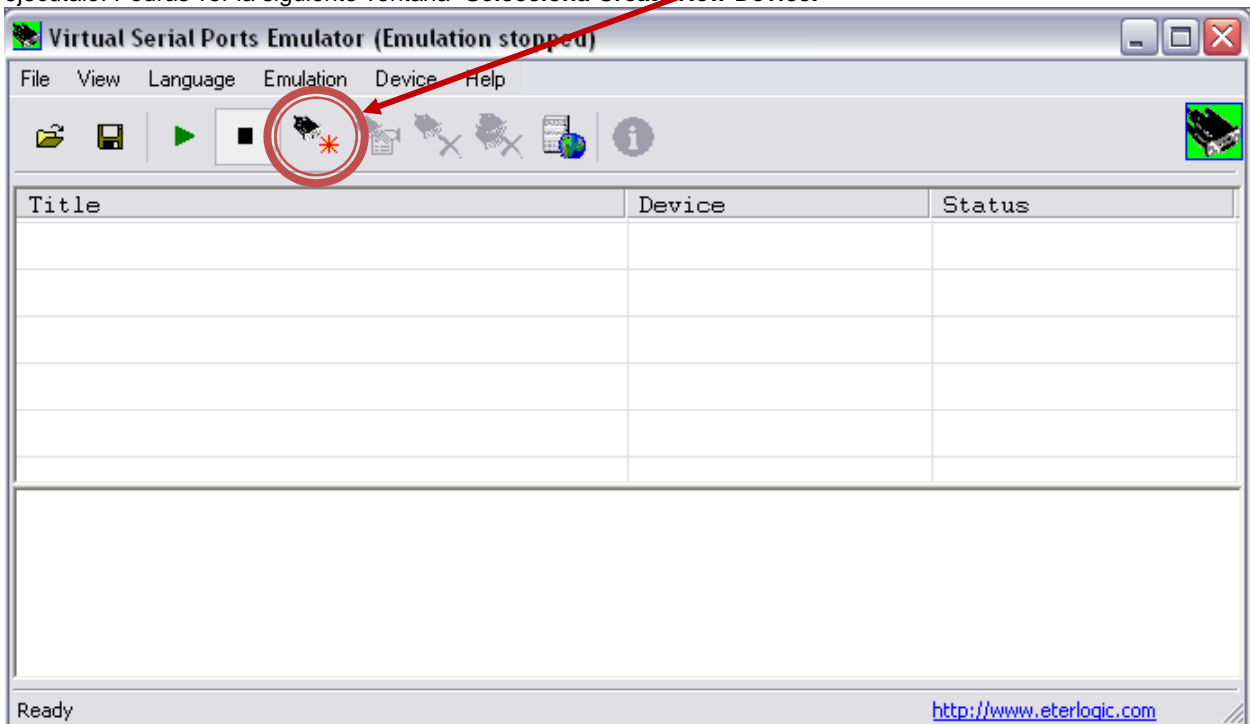
configura_puerto(cmbPuerto.Text, Convert.ToInt32(cmbBaudios.Text), Paridad, numBits, BitsParada);
}

private void btnAbrir_Click(object sender, EventArgs e)
{
    if (serialPort1 != null)
    {
        serialPort1.Open();
        MessageBox.Show("Puerto abierto");
    }
}

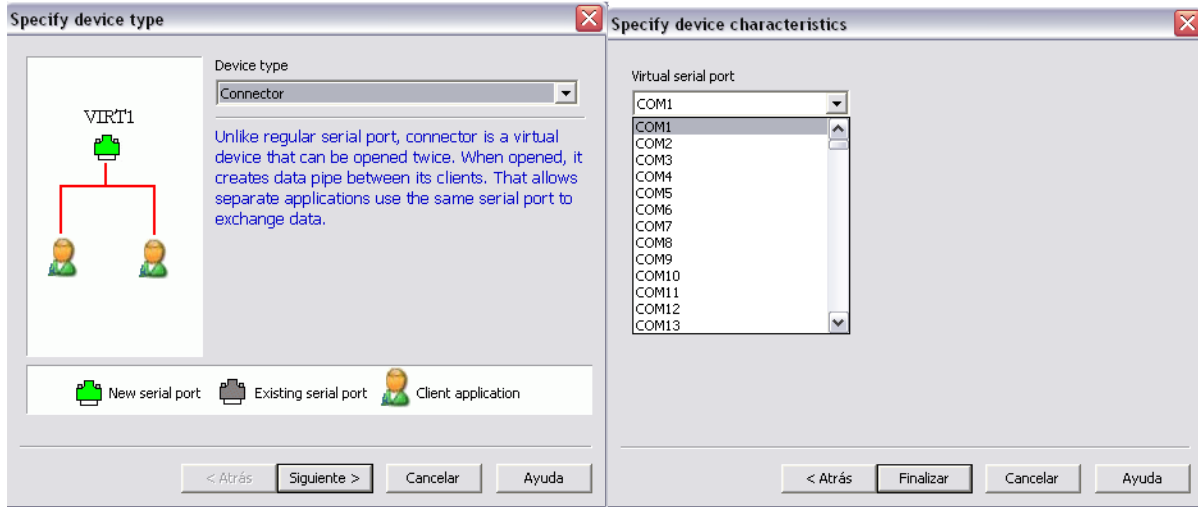
private void btnCerrar_Click(object sender, EventArgs e)
{
    if (serialPort1 != null)
    {
        if (serialPort1.IsOpen == true)
        {
            serialPort1.Close();
            MessageBox.Show("Puerto cerrado");
        }
    }
}
}
}
}

```

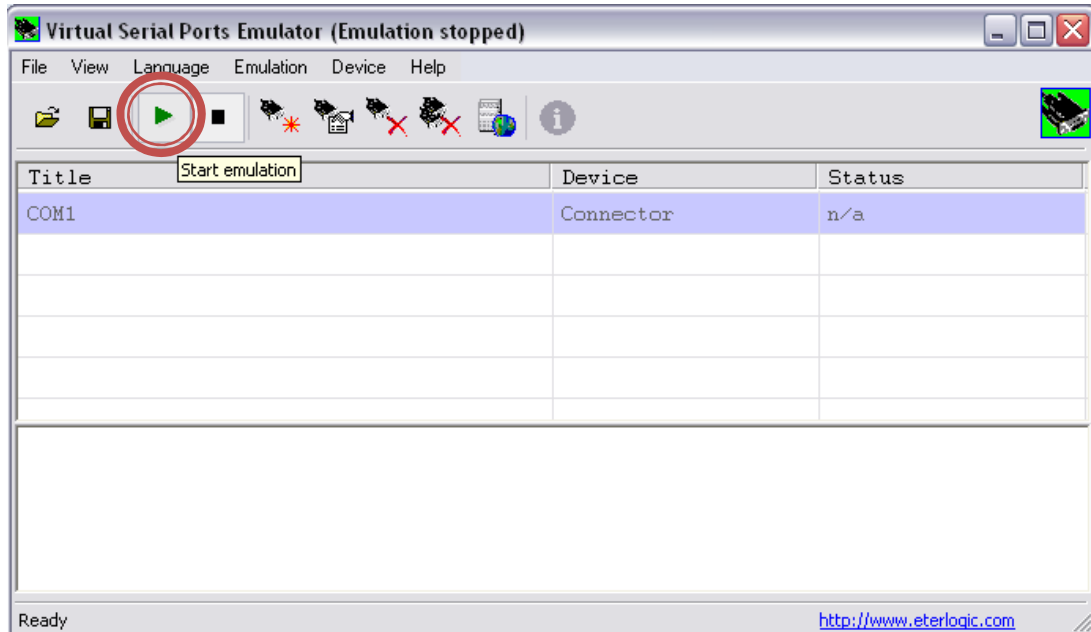
Para realizar pruebas, descarga el emulador disponible en la página de técnicas. Posteriormente instalalo y ejecútalo. Podrás ver la siguiente ventana- **Selecciona Create New Device.**



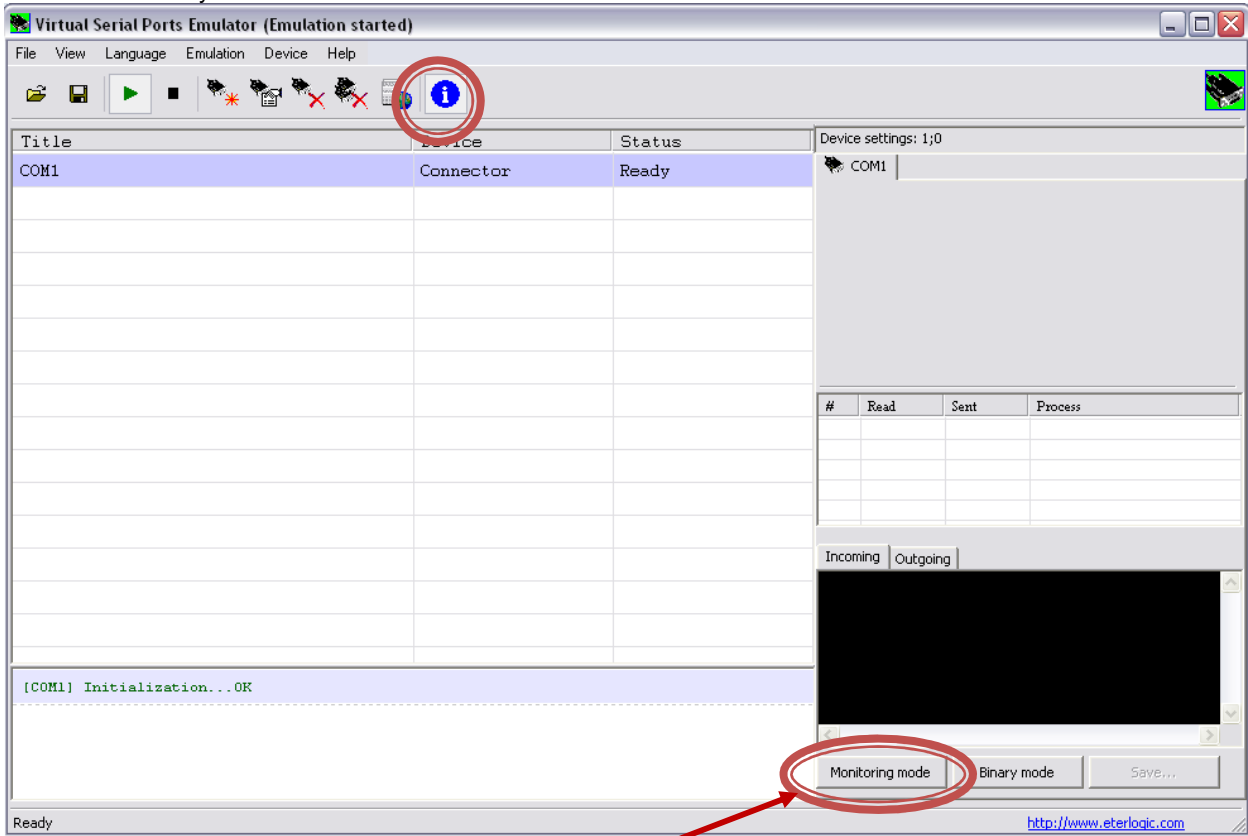
Posteriormente selecciona siguiente > Puerto a crear ("COMX") > Finalizar



Seleccione el puerto y de clic en el botón Start.

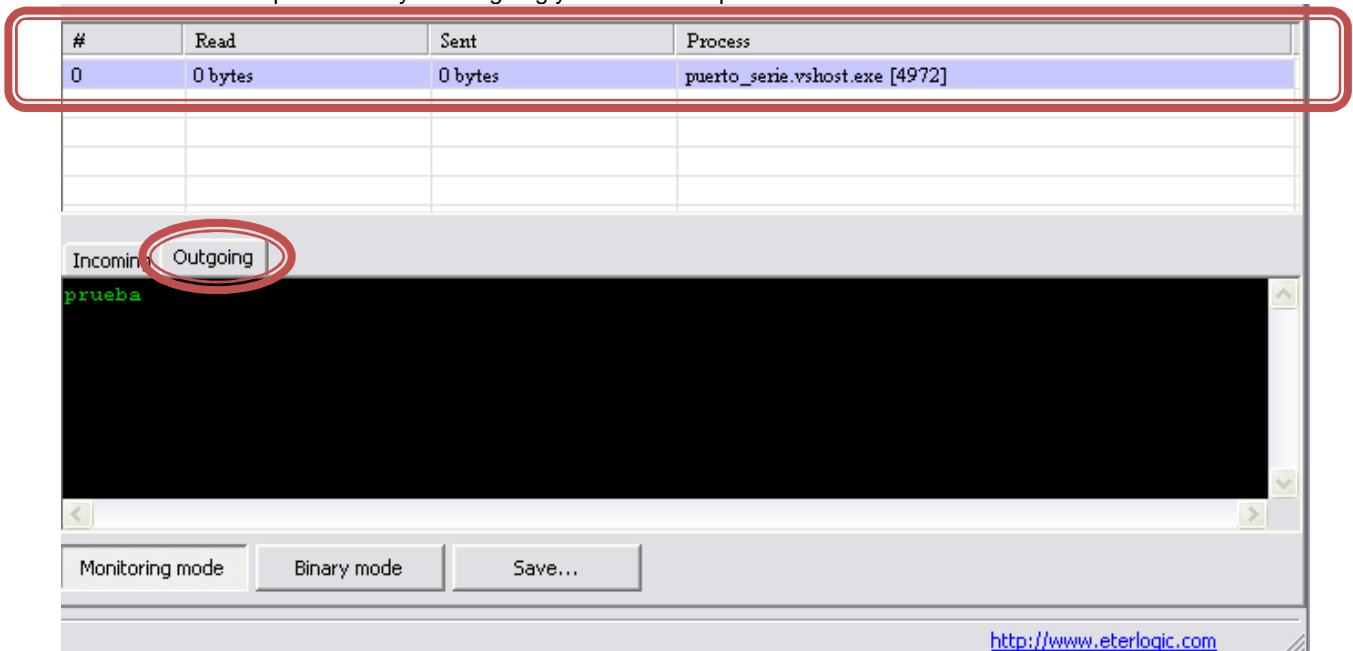


Posteriormente vaya a Menu > View > Extra information o bien seleccione el icono



Posteriormente de clic en el botón inferior **Monitoring mode**

Ejecute su programa de c#, configure el puerto para que abra el puerto creado en esta aplicación y envíe información. En la aplicación vaya a Outgoing y seleccione el proceso.



**NOTA:** Investiga como se pueden enviar datos del emulador a la aplicación.