STX80XX
Power I/O Board & PLC Cube

AN001

Diseño de aplicaciones Visual C# para Ethernet/UDP en modo PLC [1]

Autor: Ing. Boris Estudiez

Modelos Aplicables	AX, CX, DX

1 Descripción General

*La presente nota de aplicación, detalla como diseñar aplicaciones en Microsoft Visual C#, que le permitirán crear programas en su computadora para comunicarse mediante la interfaz Ethernet / UDP a los Programas Pawn que se ejecuten sobre un dispositivo **STX80XX** en Modo **PLC**.*

*Esta nota de aplicación puede serle de gran utilidad para transferir datos desde el **PLC** a una computadora conectada a la red Ethernet. En tales datos, usted puede controlar características del dispositivo **STX80XX** o puede extraer información de utilidad (sensores, registros, etc).*

*A continuación se describen dos programas, uno para enviar datos al **PLC** y otro para recibir datos desde el **PLC**.*

Nota [1]: Ejemplos disponibles para descargar en Microsoft Visual Basic .NET en página web.

2 Lecturas Recomendadas

Antes de leer este documento, recomendamos que se familiarice con el dispositivo STX80XX, el entorno StxLadder y el paquete de software SDK (**S**oftware **D**evelopment **K**it). Para ello recomendamos leer los siguientes documentos, en el orden detallado a continuación:

1. **STX80XX-GS-AX_BX_CX_DX**: Guía de Primeros Pasos especifico de su dispositivo.
2. **STX80XX-DS-AX_BX_CX_DX**: Hoja de Datos especifica de su dispositivo.
3. **STX80XX-MP-PLC-AX_CX_DX**: Manual de Programación Pawn del PLC.
4. **STXLADDER-UM**: Manual de Usuario del entorno StxLadder.

Esta nota de aplicación, se complementa con el Manual de Programación Pawn del PLC, por lo tanto es importante que primero, entienda los conceptos básicos del PLC y del lenguaje Pawn antes de continuar.

Para programar en Microsoft Visual C#, es necesario tener conocimientos básicos sobre el lenguaje de programación C# y tener instalado el entorno de desarrollo integrado (IDE).

Para facilitar su aprendizaje, hemos elaborado la siguiente guía básica, que recomendamos leer, si se programa por primera vez en C#:

1. **STX80XX-GS-CSHARP**: Guía Básica de C#.

Adicionalmente, puede adquirir libros de C# más avanzados de una librería o Internet. Muchas veces, las dudas sobre programación, ya están solucionadas y respondidas en Internet, utilizar el buscador Google (www.google.com) puede servirle de gran ayuda.

Mayor documentación puede encontrar en la pagina del producto: www.slicetex.com.

3 Requerimientos

Para esta nota de aplicación es necesario tener instalado el siguiente software:

1. **StxLadder** : Entorno de Programación para el PLC.
2. **STX80XX-SDK** : Software Development Kit.
3. **Microsoft Visual C# Express**: Versión gratuita, puede descargarse desde: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express.

Su instalación, es descripta en los documentos recomendados en la sección “Lecturas Recomendadas”.

Este documento presupone que:

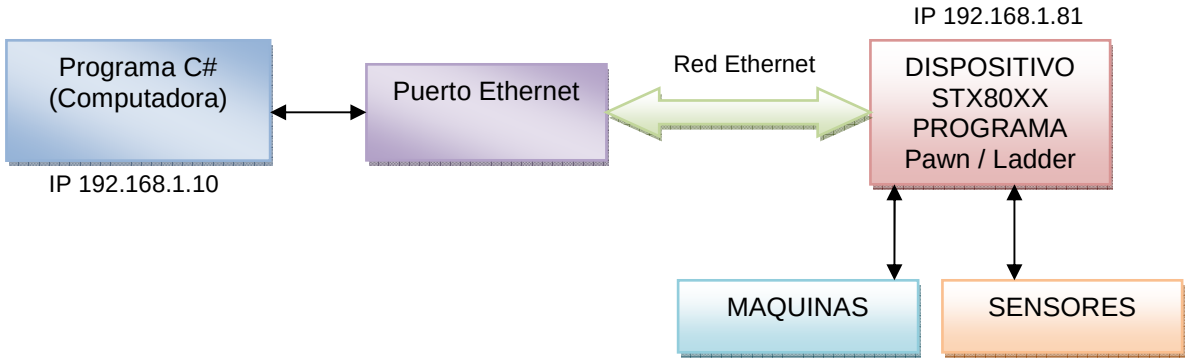
- La versión de Firmware del dispositivo es la última disponible en sitio Web.
- La versión de la librería STX8XXX.DLL es la última disponible en sitio Web.

4 Teoría de Funcionamiento

Cuando el dispositivo **STX80XX** funciona en modo **PLC**, solo es posible programarlo mediante el lenguaje **Pawn** y el lenguaje **Ladder** (no tratado en este documento).

Esto significa que si necesitamos enviar datos desde una computadora al PLC, mediante una red Ethernet, primero deberemos crear un programa que le permita recibir datos Ethernet al PLC.

Creado el programa del PLC, procedemos a diseñar una aplicación en Microsoft Visual C# para poder enviar datos por la red Ethernet al PLC.



En el anterior diagrama en bloques, se observa que el usuario, realiza su programa en C#, lo ejecuta en una computadora y luego comanda a través de la interfaz Ethernet / UDP el dispositivo STX80XX que está ejecutando un programa Pawn en modo PLC.

En el programa del PLC, se reciben a través de la red Ethernet los paquetes UDP, y posteriormente se los procesan. Una vez procesados, el programa decide por ejemplo controlar Maquinas a través de las salidas DOUTx y leer valores de Sensores varios mediante entradas analógicas. Luego envía el resultado a la computadora mediante una transmisión Ethernet / UDP.

NOTA: *Si desea utilizar una computadora para adquirir datos y realizar tareas de control mediante una computadora de forma simple, puede cambiar el modo de funcionamiento del dispositivo a* ***Modo DAQ****, el cual permite el control total del mismo a través de la red Ethernet. Consulte el manual de usuario Modo DAQ, para más información (****STX80XX-UM-DAQ-AX_BX****).*

5 Arquitectura de la Comunicación

El dispositivo STX80XX tiene una forma particular de comunicarse mediante una red Ethernet. La comunicación de datos, se realiza con el protocolo UDP (User Datagram Protocol).

La forma de recibir y transmitir datos o paquetes UDP en modo PLC, difiere. A continuación, explicamos la teoría de comunicación del dispositivo.

5.1 Recepción de datos UDP

Internamente, cuando el dispositivo recibe datos UDP se los considera “comandos”, que forman parte del protocolo **CSP** (**C**ommand **S**erver **P**rotocol) diseñado por Slicetex Electronics.

El protocolo **CSP**, tiene las siguientes características principales:

- Permite comprobar si los datos llegaron a destino (al dispositivo sTX80XX).
- Orientado a transmitir comandos y recibir sus resultados.
- Permite conocer el estado del comando enviado al dispositivo.
- Permite establecer una clave (password) de 32-bit para asegurar que solo el usuario que conozca la clave del dispositivo pueda ejecutar los comandos enviados.

*Entonces, para enviar un dato UDP al dispositivo STX80XX cuando funcione en modo PLC, deberemos utilizar el protocolo **CSP**, sobre el cual enviaremos un comando con los datos a transferir desde la computadora al PLC.*

Cuando usted programe aplicaciones en Visual C#, podrá enviar datos de forma simple y natural, ya que el protocolo CSP esta implementado en la librería que se distribuye. Pero es importante saberlo, para poder enviar los datos correctamente y/o en caso de utilizar algún otro lenguaje de programación.

Para poder recibir comandos y ejecutarlos, el dispositivo STX80XX, tiene asignada una dirección IP (por defecto 192.168.1.81) y un puerto UDP (4950) donde escucha la llegada comandos.

Más información sobre el protocolo CSP, en el manual de usuario del modo DAQ.

5.2 Transmisión de datos UDP

La transmisión de datos desde el PLC a una computadora mediante la red Ethernet, se realiza con el protocolo UDP en forma pura. Es decir, los datos son enviados sin ningún protocolo intermedio.

Nota:

Al protocolo UDP se lo denomina sin conexión. Esto quiere decir que no se garantiza que un paquete llegue a destino o que no contenga errores al llegar. En una red local Ethernet es poco probable la existencia de errores. Si sus datos son importantes, deberá crear un pequeño protocolo (entre el PLC y el dispositivo de destino) para que en ambos extremos se garantice que los datos llegaron y adicionalmente que no contienen errores. Esto debería tenerse en cuenta solo para la transmisión de datos ya que para la recepción se utiliza el protocolo CSP que intenta resolver algunos problemas del UDP.

6 Diseño de Aplicación para Transmitir Datos

Siguiendo la pseudo-filosofía de Slicetex: "Aprenda practicando"… en esta sección mostraremos un procedimiento practico para crear el primer programa para transmitir datos en C# a nuestro PLC.

El programa se comunicara con el dispositivo STX80XX y enviara dos bytes que serán leídos por el PLC, programado previamente utilizando StxLadder.

El primer byte le indicará al PLC que debe hacer con el RELAY1 o salida DOUT1. Si el byte tiene un valor de "1", activa la salida. Si su valor es "0", desactiva la salida.

El segundo byte se envía con valor cualquiera para fines didácticos.

El programa en C#, se diseñara con dos botones, un botón llamado "ACTIVAR" que activará DOUT1 al hacer click sobre él. Otro botón, llamado "DESACTIVAR" desactivará DOUT1. Cuando se haga click en alguno de estos botones, se transmitirán los bytes de datos al dispositivo.

Adjunto a este documento, podrá encontrar el programa completo, listo para compilar y ejecutar en Microsoft Visual C# 2005. El proyecto se llama "**Prueba1**".

Nota: Ejemplo disponible para descargar en Microsoft Visual Basic .Net también.

Asegúrese de entender bien el procedimiento mostrado, ya que el resto de los ejemplos descriptos en este documento parten de la suposición que usted entiende el proceso para enviar comandos al dispositivo STX80XX mediante C#.

6.1 Requerimientos Previos

Para poder enviar comandos al dispositivo es necesario tener instalado el paquete de software STX80XX-SDK (Software Development Kit) y Microsoft Visual C# Express.

Se recomienda leer los siguientes documentos antes de continuar:

1. **STX80XX-GS-AX_BX_CX_DX** : Guía de Primeros Pasos del dispositivo.
2. **STX80XX-GS-CSHARP**: Guía Básica de C#.

6.2 Librería STX8XXX.DLL

La librería STX8XX.DLL es un archivo DLL (Dynamic Link-Library), que contiene la interfaz necesaria para que usted pueda comunicarse con el dispositivo STX80XX desde sus programas. *Esta librería debe ser enlazada o referenciada desde cada programa que usted diseñe.*

La librería **STX8XXX.DLL** se encuentra dentro del directorio donde usted instalo el SDK (Software Development Kit), en la carpeta:

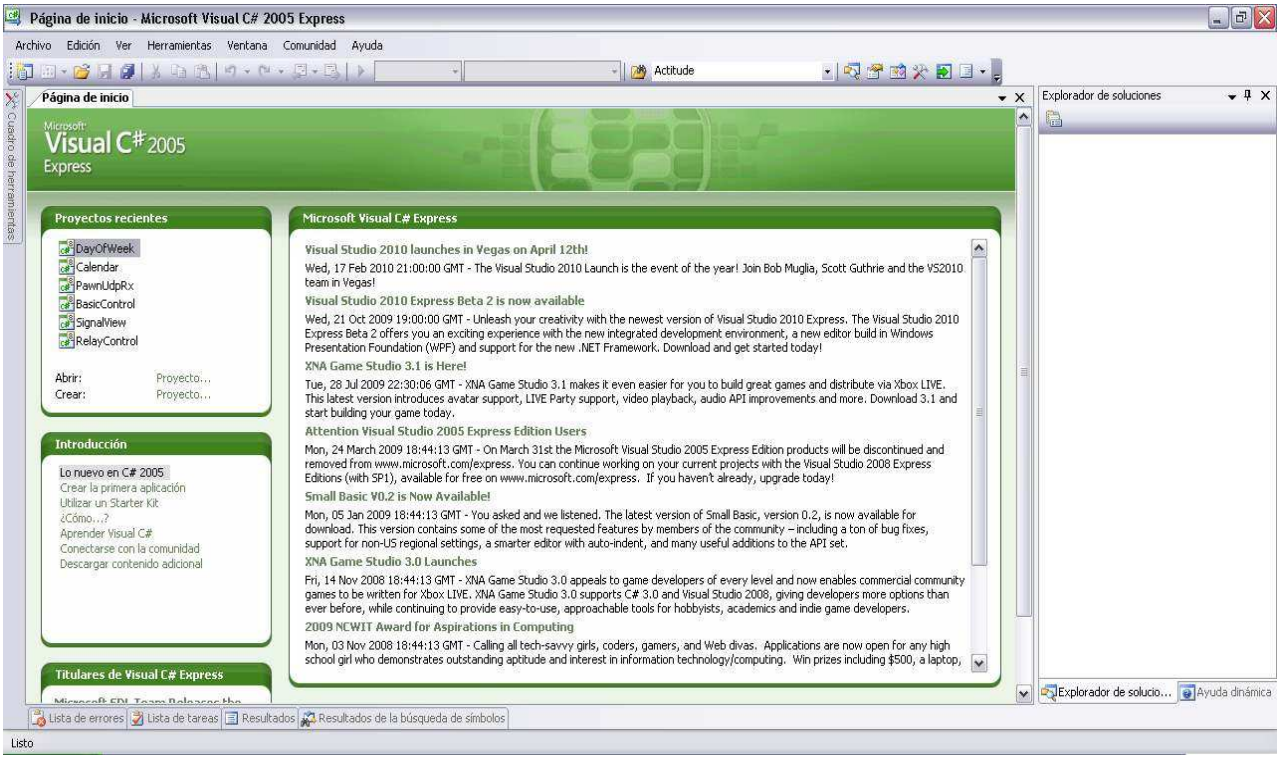
visual_cs\libs\stx8xxx

6.3 Diseñar el Programa

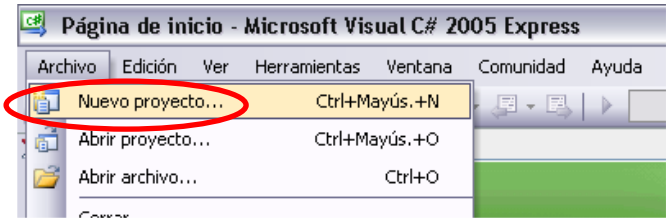
Lo primero que debe hacerse al crear un programa con interfaz grafica, es diseñar la interfaz visual. Para ello abriremos entorno de desarrollo “Microsoft Visual C# Express”, haciendo doble click en el icono del programa:

Nota: En este documento se utiliza “Microsoft Visual C# Express 2005”, usted puede utilizar cualquier otra versión, el concepto es el mismo.

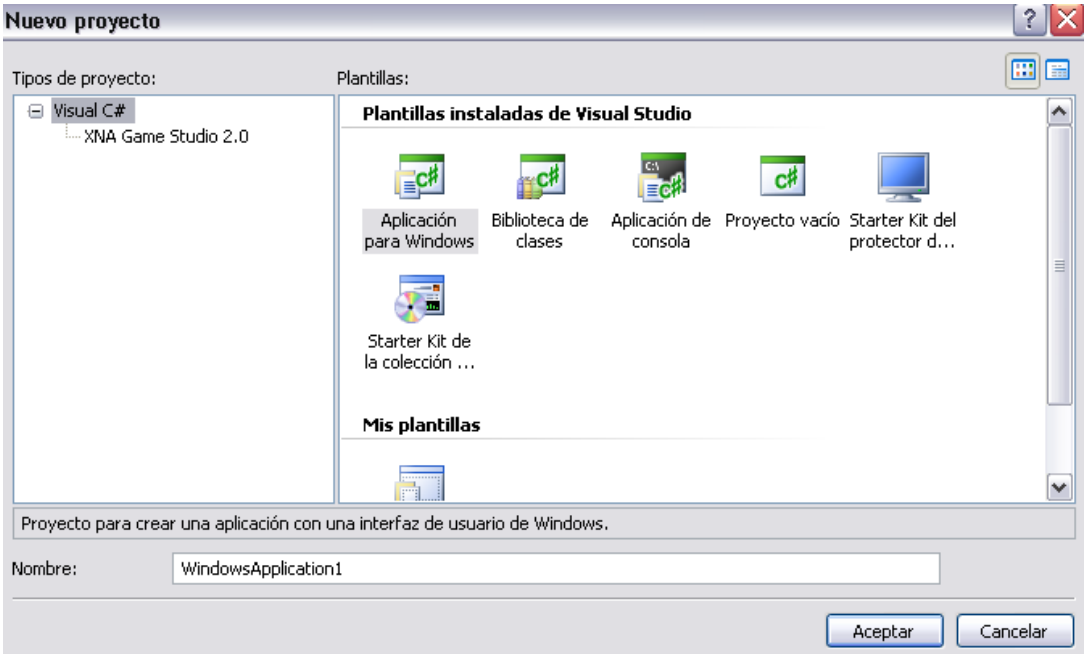
La siguiente pantalla aparecerá al abrir el entorno:



Para crear un nuevo programa, seleccionamos el menu “Archivo” y luego hacemos click en “Nuevo proyecto”:

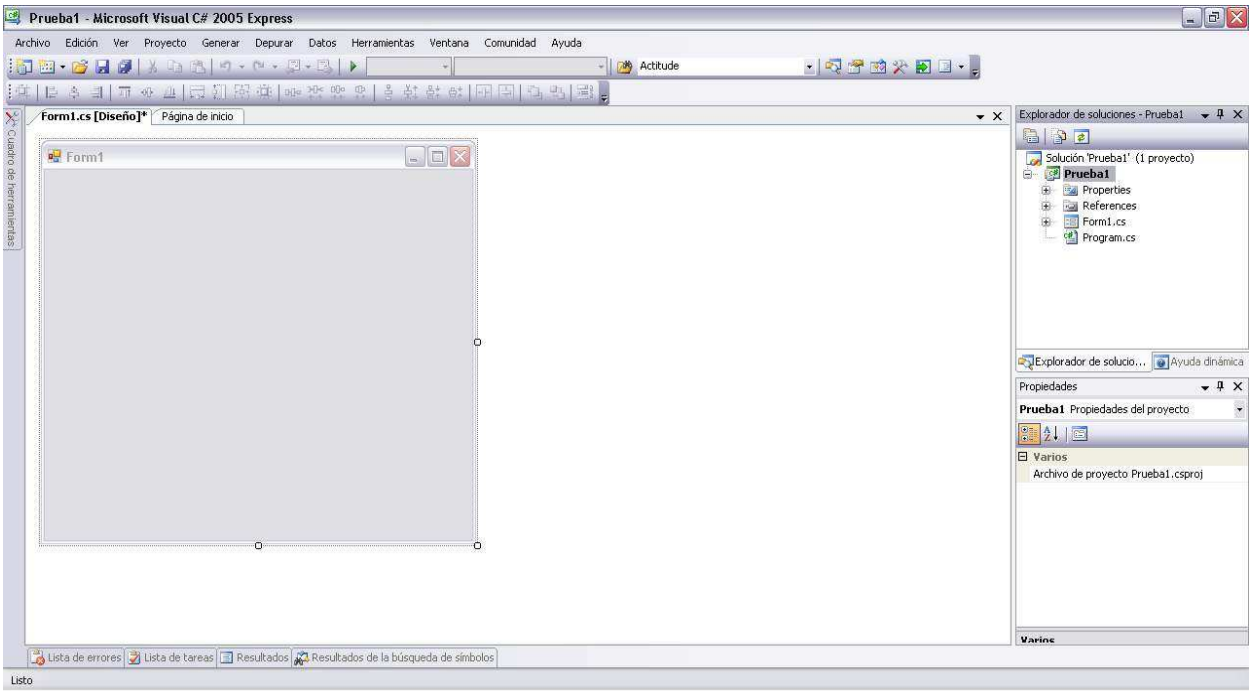


Aparecerá la siguiente ventana:



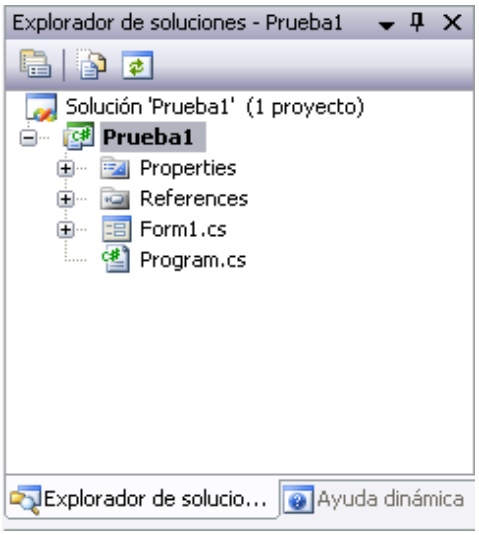
En tipos de proyecto elegimos “Visual C”, en plantillas seleccionamos “Aplicación para Windows”, en nombre escribimos “Prueba1”, el mismo será utilizado como nombre de nuestro programa.

Finalmente, hacemos click en “Aceptar” y obtenemos la siguiente pantalla con nuestro programa:

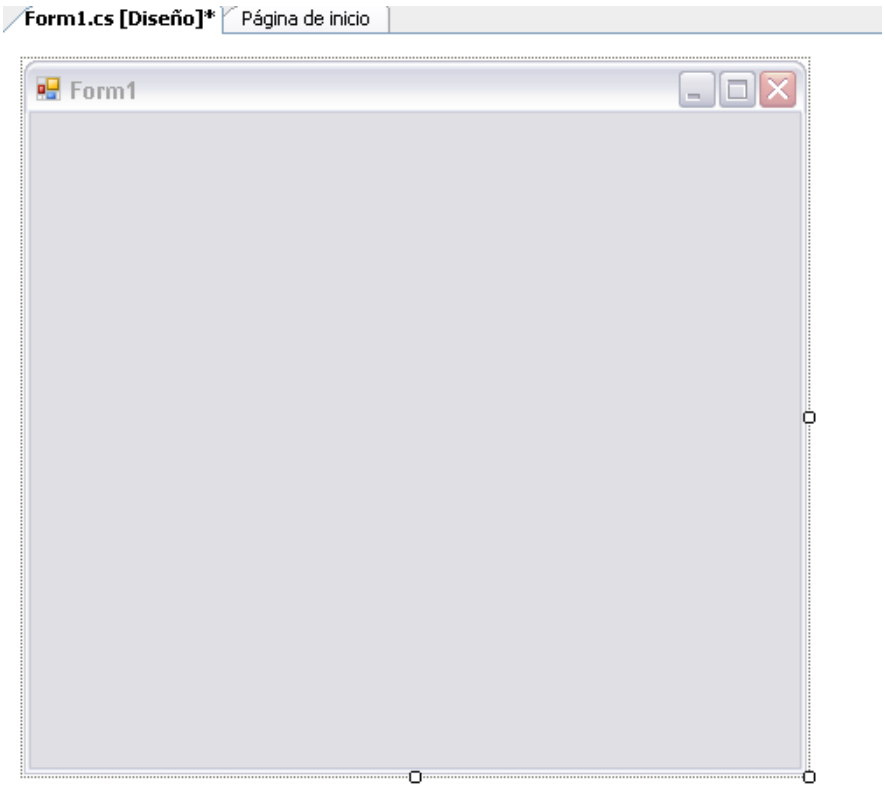


En la pantalla anterior, hay cuatro secciones que debe identificar:

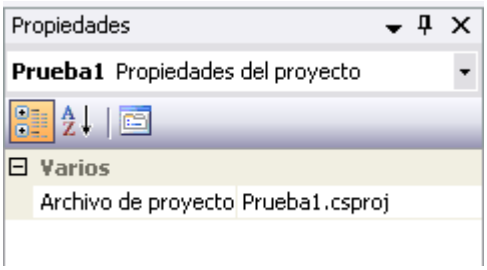
1 - Explorador de Soluciones: Aquí se listan todos los archivos del proyecto.



2 - Formulario: Aquí esta la ventana principal de nuestro programa, donde usted podrá colocar botones, cajas de texto, etc.

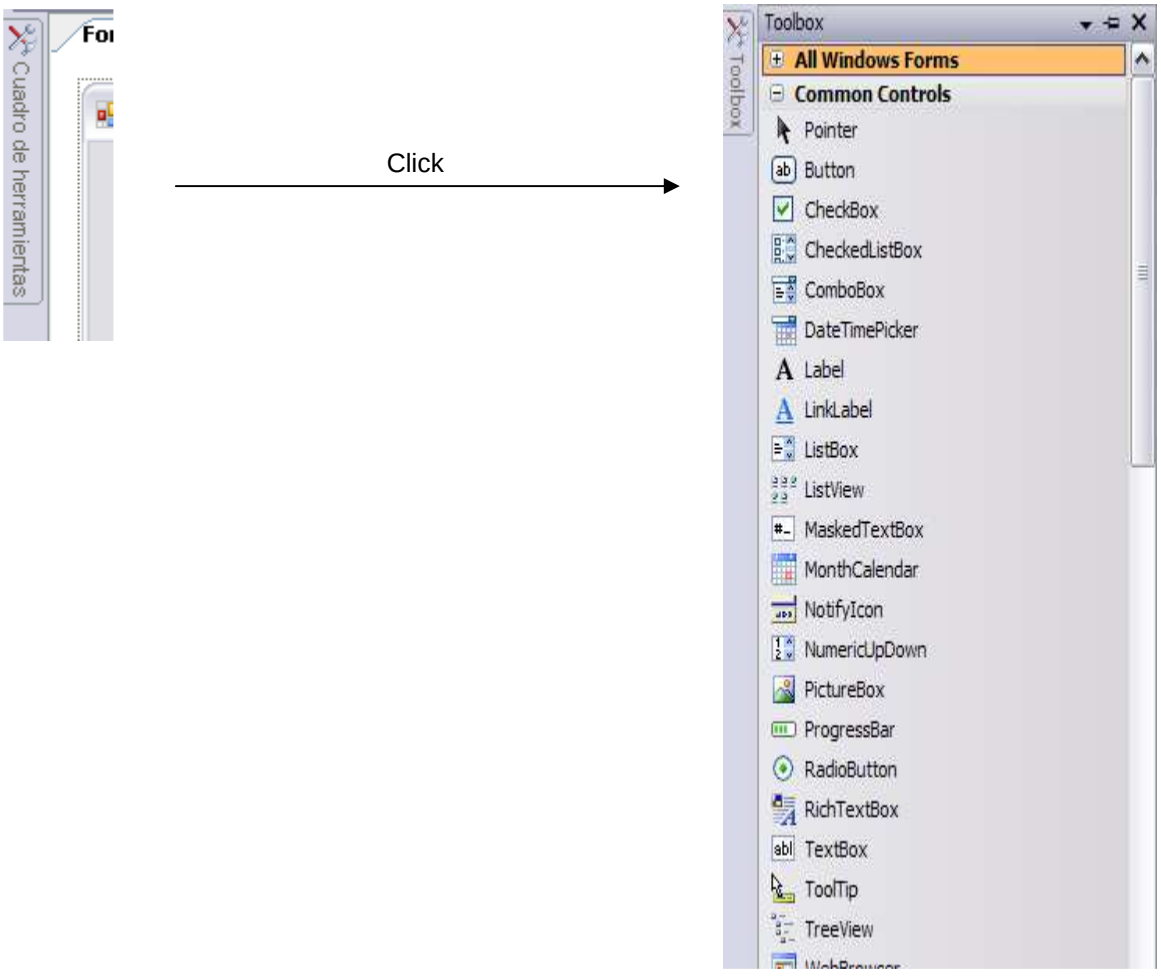


3 – Ventana de Propiedades: Aquí aparecerán las propiedades de cada objeto grafico que coloque en el formulario (ancho de ventana, tipo de letra de un cuadro de texto, etc.).



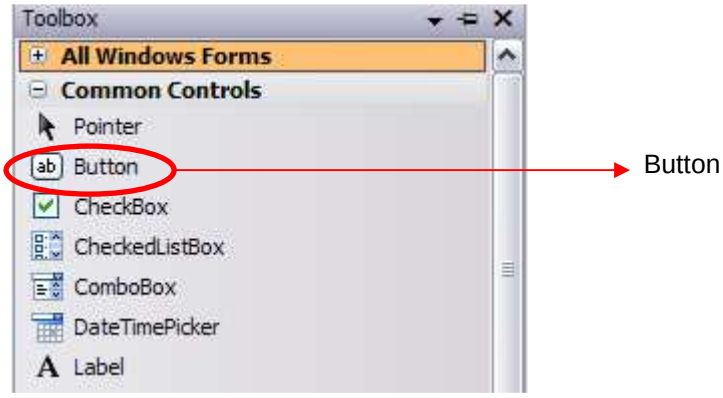
4 – Cuadro de Herramientas: Contiene los objetos que usted puede utilizar en su formulario.

Click en la pestaña que dice “Cuadro de Herramientas” al costado izquierdo de la pantalla:

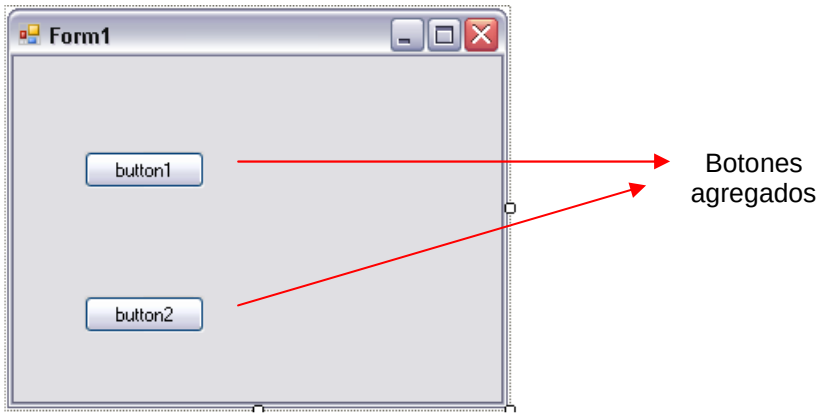


El próximo paso será colocar dos botones en el formulario:

Para ello vamos al “Cuadro de Herramienta” y seleccionamos “Button”, que es el botón:

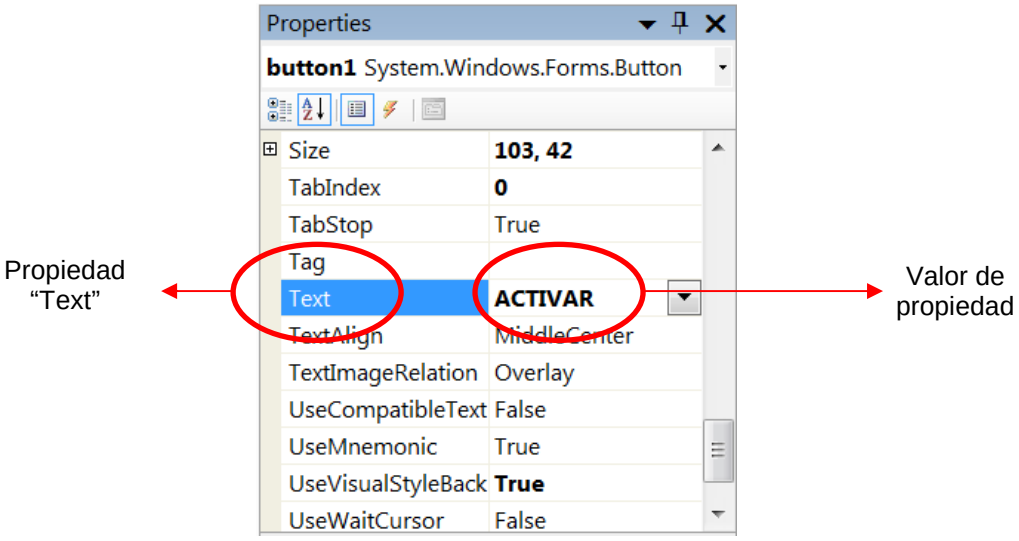


Hacemos click en “Button” y mantenemos apretado el Mouse y lo arrastramos al formulario. Repetimos el procedimiento una vez más, para agregar dos botones, y nuestro formulario debería verse de la sig. forma:

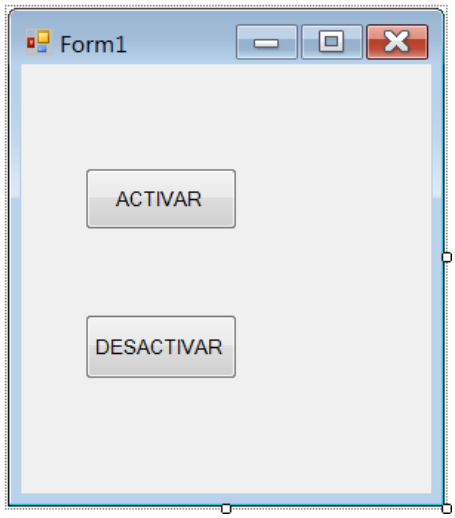


Ahora, cambiaremos los nombres mostrados de los objetos en el formulario, para mejor visualización:

Click en el "botton1" y luego desde la "Ventanas de Propiedades", buscamos la propiedad "Text" y su valor asignado, lo cambiamos por "CERRAR",



Procedemos de la misma manera para el "botton2", pero esta vez, cambiamos el valor de "Text" a "ACTIVAR". La pantalla del programa lucirá de la siguiente forma:

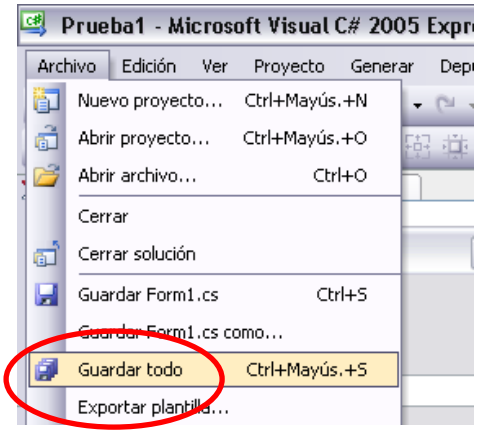


Nota: Cambiar el valor de la propiedad "Text" de un objeto, solo cambia el texto mostrado en pantalla. Para cambiar el nombre de un objeto, hay que cambiar la propiedad "Name".

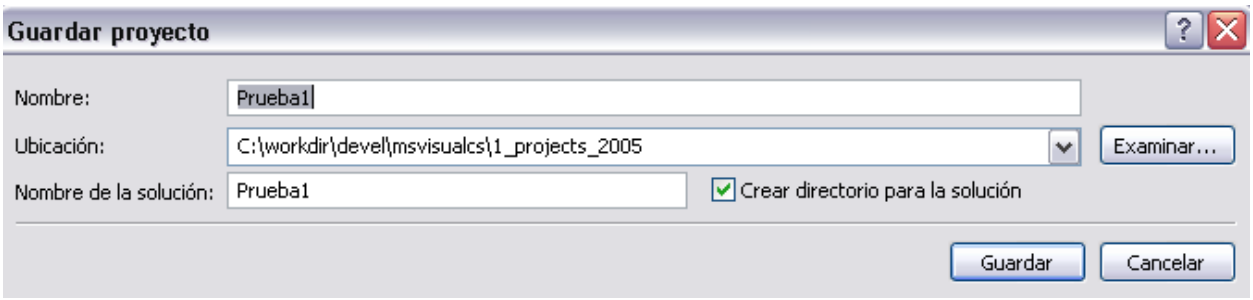
Luego, cuando hagamos click en "ACTIVAR", el RELAY1/DOUT1 se activará. Y cuando hagamos click en "DESACTIVAR", el RELAY1/DOUT1 se desactivará.

6.4 Guardar el Proyecto

Para guardar el proyecto, haga click en “Archivo” y luego en “Guardar todo”:



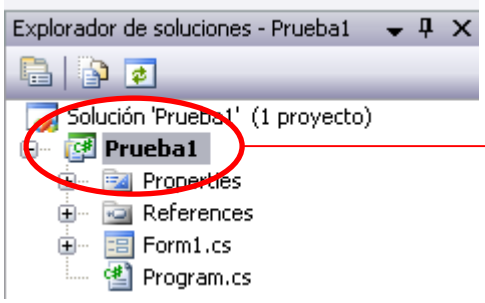
Le aparecerá la siguiente pantalla:



Elija la ubicación o directorio donde quiera guardar el proyecto (o solución en términos de Visual C#) y haga click en “Guardar”. En este punto su proyecto fue guardado en disco.

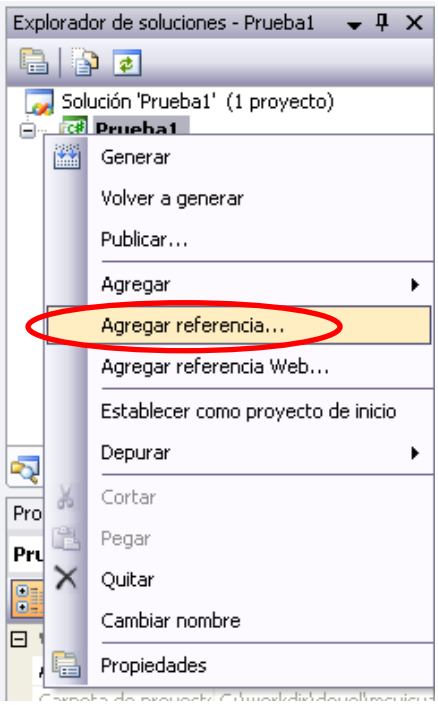
6.5 Referenciar Librería STX8XXX.DLL

Ahora vamos a enlazar o referenciar el archivo STX8XXX.DLL que contiene la librería para operar el dispositivo STX80XX. Para ello nos dirigimos al explorador de soluciones “Explorador de Soluciones”, y hacemos click derecho en “Prueba1”:

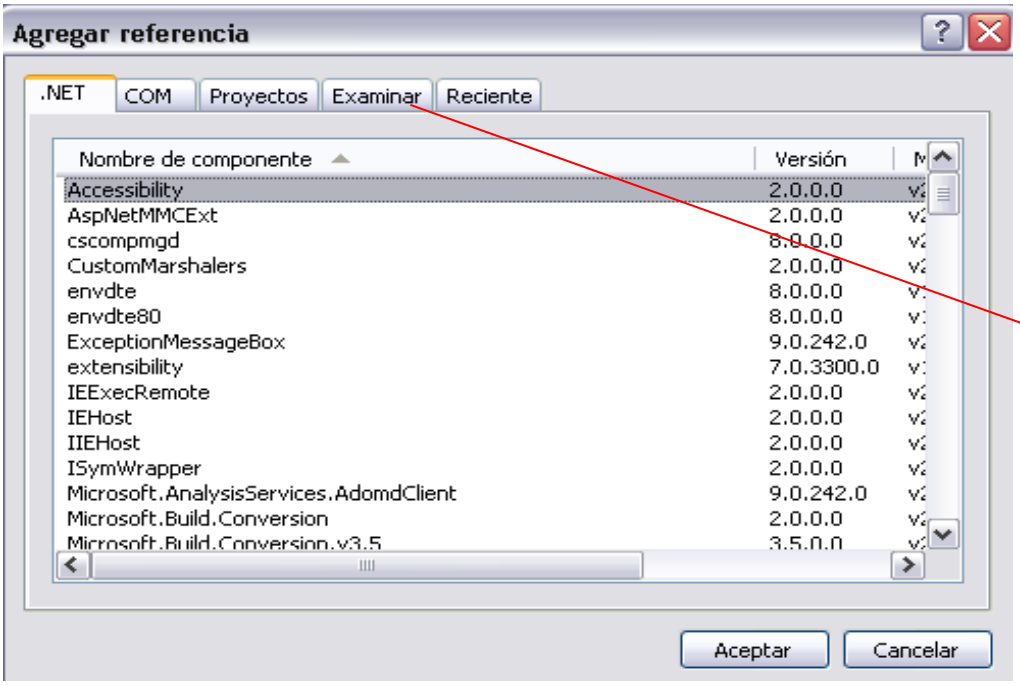


Click derecho en este archivo de proyecto “Prueba1”

Al hacer click derecho, el siguiente menú se desplegara, en donde haremos click en el item “Agregar referencia…”:



La siguiente pantalla aparecerá, y haremos click en la pestaña “Examinar”:

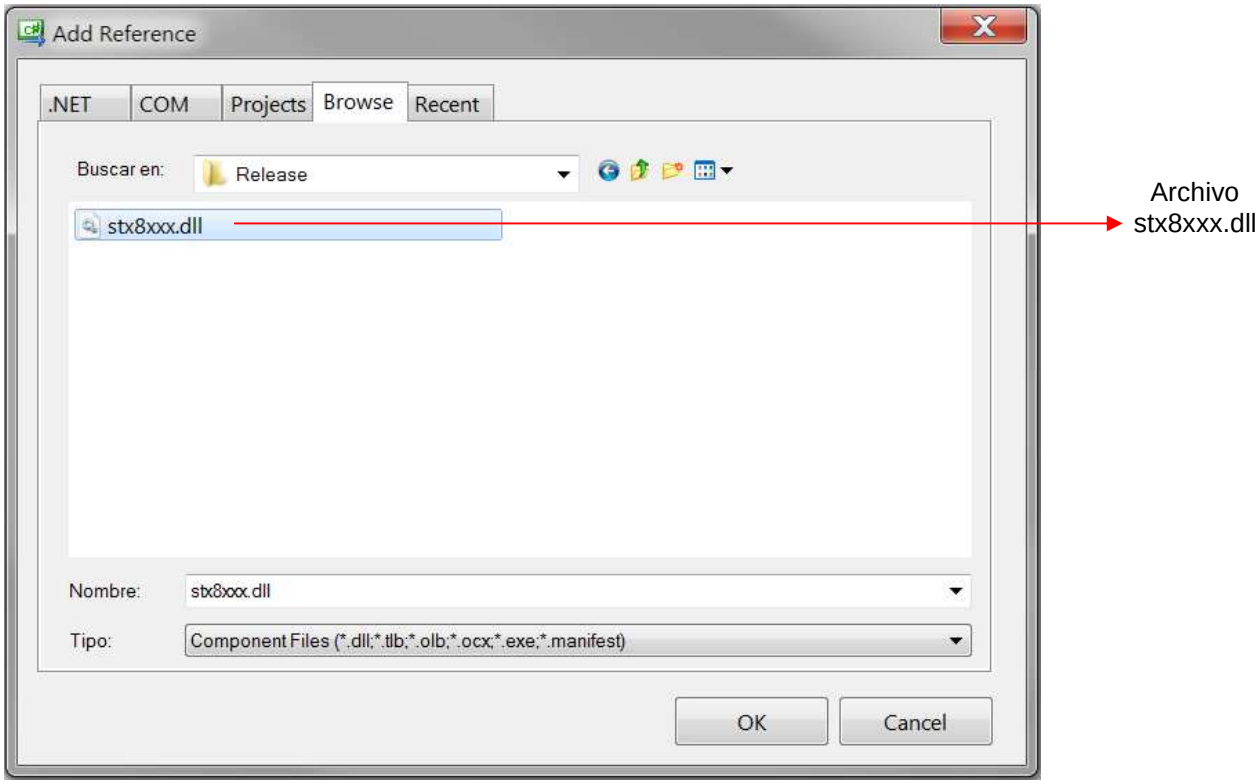


Click en pestaña “Examinar”

En la pestaña “Examinar” buscaremos el archivo “stx8xxx.dll”, que es la librería. Para ello nos deberemos deslazar al directorio donde se encuentra el software SDK (Software Development Kit) y bajo el directorio:

visual_cs\libs\stx8xxx

Seleccionaremos el archivo “stx8xxx.dll” como se muestra a continuación:

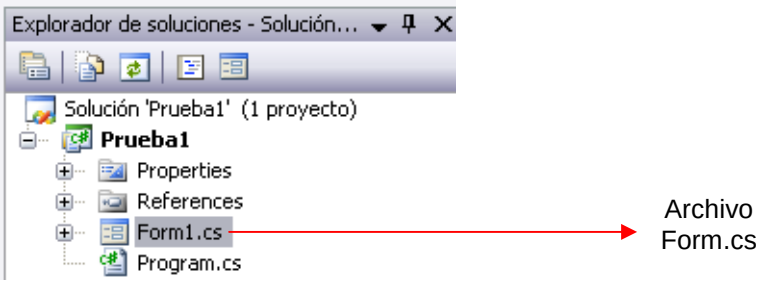


Luego click en “Aceptar” y listo, la librería fue agregada a nuestro proyecto.

6.6 Inicializar la Librería

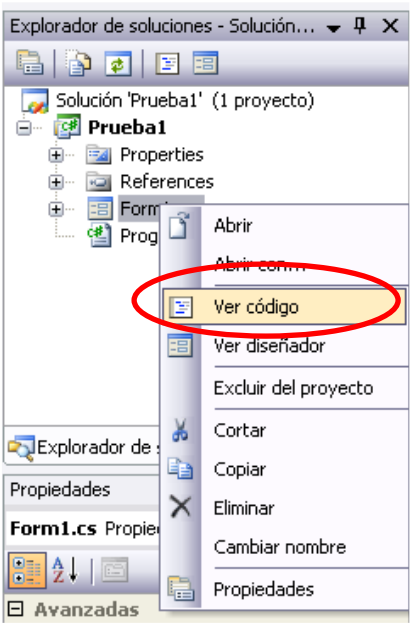
Ahora procederemos a agregar código en C#, para inicializar o poner a punto la librería.

Primero, vamos a agregarla al especio de nombres de nuestro programa, utilizando la directiva “using”. Para ello vamos al explorador de soluciones, y hacemos click en el archivo de nuestro formulario principal “Form1”:

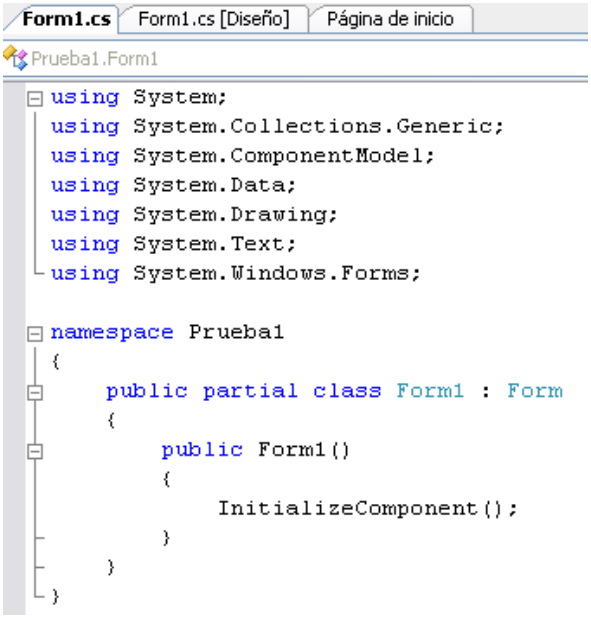


El archivo Form.cs contiene el código C# de nuestra ventana o formulario principal del programa.

Hacemos click derecho y luego seleccionamos “Ver código”:



Se debería abrir la siguiente ventana:



La misma muestra el código en C# que ejecuta nuestro programa actualmente. Para agregar nuestra librería al espacio de nombres del programa, utilizaremos la directiva “**using**”, agregando “**using stx8xxx;**” al archivo Form.cs y el código nos quedaría como sigue:

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

using stx8xxx;

namespace Prueba1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Directiva using agregada.

El próximo paso es crear un objeto llamado **PioBoard**, el cual contendrá todos los métodos y clases para acceder al dispositivo STX80XX. El nombre **PioBoard** significa “Power I/O Board”, pero usted puede nombrarlo con cualquier otro nombre.

Dentro de la clase “Form1”, creamos una variable global que será nuestro objeto del tipo **Stx8xxx**, y se llamará **PioBoard**.

```
Stx8xxx PioBoard;
```

El codigo resultante quedaria de la siguiente forma (se omiten las directivas “using” por simplicidad):

```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Ahora vamos a inicializar la librería, esto se hace en el constructor de la clase Form1 y con el siguiente codigo:

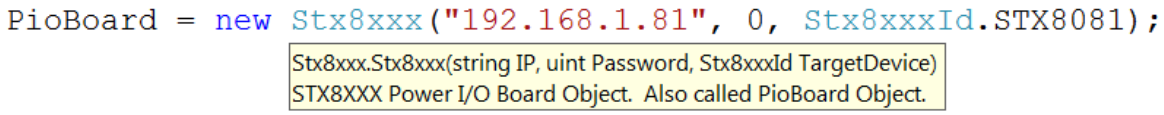
```
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081));
```

Donde asignamos a PioBoard un objeto proveniente de la clase Stx8xxx, al cual se le pasó la dirección IP del dispositivo “192.168.1.81” como primer argumento y como segundo argumento el password a

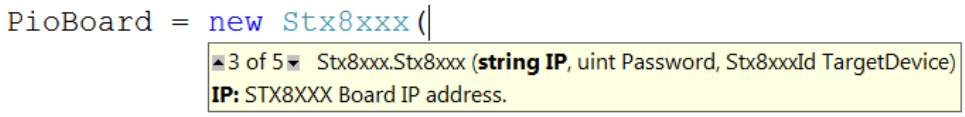
utilizar para enviar comandos, en este caso 0. El tercer argumento indica el dispositivo utilizado, por ejemplo la STX8081.

Para facilitar la codificación, la librería STX8XXX.DLL tiene documentación en línea, la cual está disponible si ponemos el puntero del Mouse sobre algún nombre.

Por ejemplo, al poner el Mouse sobre Stx8xxx, aparece el siguiente cartel, que nos muestra que es un objeto “Power I/O Board”.



Si queremos información sobre sus argumentos, al escribir la línea, luego del “(“ nos aparecerá un cartel con las posibles formas de inicializar la librería, apretamos la tecla “DOWN” y elegimos la opción “3”, apareciendo un cartel con la información del argumento:



En ingles, nos dice que el primer argumento es del tipo “string” llamado IP y significa “Dirección IP del dispositivo.

Finalmente, el código resultante es:

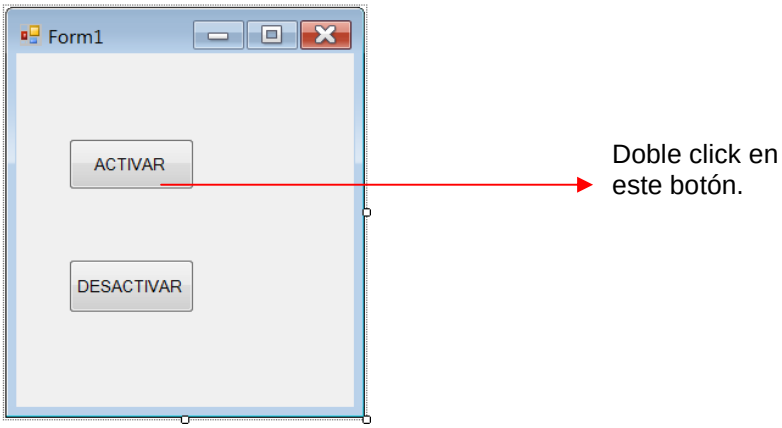
```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();

            PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081));
        }
    }
}
```

En este punto, la librería fue creada e inicializada. Notar que la dirección IP, como el valor del password, son valores que vienen por defecto en el dispositivo, pero usted puede cambiarlos con el programa StxLadder desde el menú “PLC / Configurar de PLC” (lea documento **STXLADDER-PC**).

6.7 Enviando Datos

Nuestra próxima acción consistiría en enviar un dato al dispositivo que permita activar la salida DOUT1 / RELAY1 cada vez que hagamos un click en el botón “ACTIVAR” de la ventana principal del programa.

Para lograrlo, deberemos asociar un evento “Click” al botón del formulario. Esto se puede hacer haciendo doble click sobre el botón “ACTIVAR” del formulario principal, en la vista de diseño:

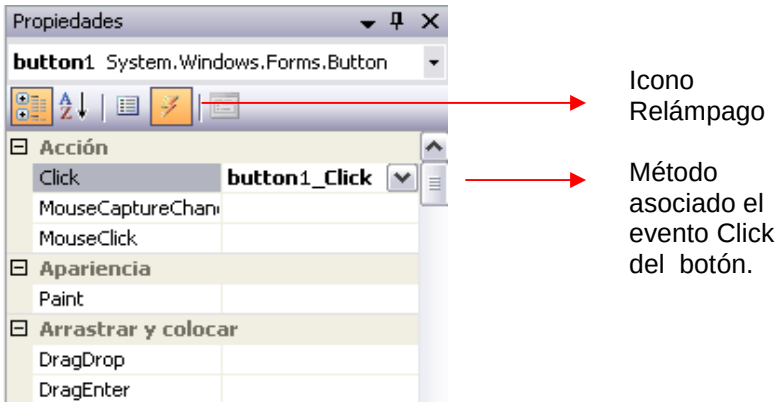


Automáticamente, se nos abrirá la ventana de código con el siguiente método creado, que representa al evento “Click” del botón:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Notar que el objeto botón “ACTIVAR” se llama “button1”, por lo tanto el método creado automáticamente por el entorno Visual C#, comienza con el nombre del objeto: “button1”.

Para visualizar los eventos asociados a un objeto, ir a la ventana de propiedades y hacer click en el icono “relámpago”, donde se nos mostrara lo siguiente:



Ahora dentro del método del evento creado (`button1_Click`), debemos escribir el comando para enviar datos UDP y activar la salida DOUT1 del PLC. Entonces escribimos las siguientes líneas:

```
// Variable para retorno de estado del PLC.
UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

// Crear array de bytes a enviar (dos bytes).
byte[] DataBytes = new byte[2];

// Especificar el primer byte el valor "1", para activar DOUT1.
DataBytes[0] = (byte) 1;

// Especificar el segundo byte un valor cualquiera.
DataBytes[1] = (byte) 55;

// Enviar bytes UDP al script con el metodo "Send".
PioBoard.Cmd.Udp.Send(DataBytes, 2, out OnUdpRxStat);
```

Con las líneas anteriores, ya es posible enviar los datos al PLC… ¿ simple verdad ?.

La primera línea crea una variable `OnUdpRxStat` del tipo enumeración `UdpRxCmdStat` que nos permitirá conocer si el PLC pudo procesar los datos enviados (si estaba habilitado, si tenía su buffer de recepción lleno, etc) cuando llamemos al método Send(). Para más información, ver Enumeración UdpRxCmdStat, página 47.

La segunda línea crear un array de 2 bytes, llamado DataBytes. Dichos bytes, serán enviados al PLC.

En la tercera línea se le asigna el valor “1” al byte DataBytes[0], que al ser recibido por el programa del PLC, cerrará el RELAY1.

En la cuarta línea se le asigna el valor “55” al byte DataBytes[1], solo con fines de prueba y no tendrá efecto en el PLC.

Finalmente, la quinta línea envía el array de datos DataBytes[] al PLC. En el primer argumento, especificamos el array a enviar, en el segundo argumento la cantidad de bytes a enviar, y en el tercer argumento la variable `OnUdpRxStat,` que almacenará el estado del comando ejecutado en el PLC. El método Send(), se explica en detalle en la pagina 43.

Nuestro próximo paso consiste en enviar un dato al PLC que permita desactivar la DOUT1 cada vez que hagamos un click en el botón “DESACTIVAR” de la ventana principal del programa.

Para ello procedemos igual que con el botón “DESACTIVAR”, y hacemos doble-click en el botón “DESACTIVAR” para crear el evento:

```
private void button2_Click(object sender, EventArgs e)
{

}
```

Ahora dentro del método del evento creado (`button2_Click`), debemos escribir el comando para enviar datos UDP y desactivar la salida DOUT1 del PLC. Entonces escribimos las siguientes líneas:

```
// Variable para retorno de estado del PLC.
UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

// Crear array de bytes a enviar (dos bytes).
byte[] DataBytes = new byte[2];

// Especificar el primer byte el valor "0", para desactivar DOUT1.
DataBytes[0] = (byte) 0;

// Especificar el segundo byte un valor cualquiera.
DataBytes[1] = (byte) 66;

// Enviar bytes UDP al script con el metodo "Send".
PioBoard.Cmd.Udp.Send(DataBytes, 2, out OnUdpRxStat);
```

El código es similar al usado para el botón “ACTIVAR”, excepto en las líneas:

```
// Especificar el primer byte el valor "0", para abrir RELAY1.
DataBytes[0] = (byte) 0;

// Especificar el segundo byte un valor cualquiera.
DataBytes[1] = (byte) 66;
```

Donde el byte DataBytes[0] se le asigna el valor “0” para activar la DOUT1 y al byte DataBytes[1] se le asigna el valor “66” con propósitos de prueba.

6.8 Compilando el Programa

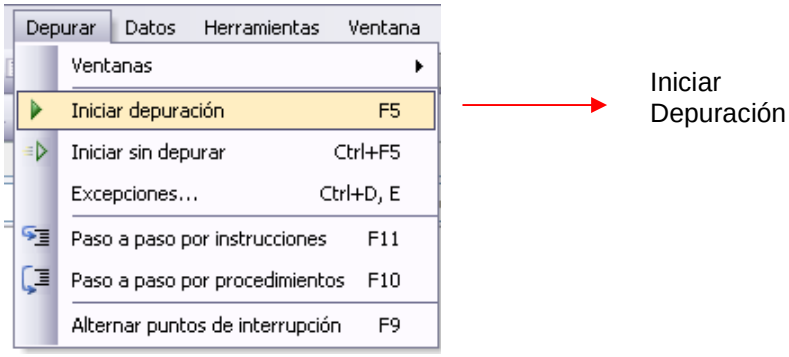
Para compilar el programa C# y luego ejecutarlo, hay dos alternativas, generar una versión “Debug”, la cual es optima para depurar programas, o generar una versión “Release”, que genera el archivo binario .EXE del programa, optimo para utilizarlo como una aplicación más de computadora.

El código completo del programa, es el siguiente:

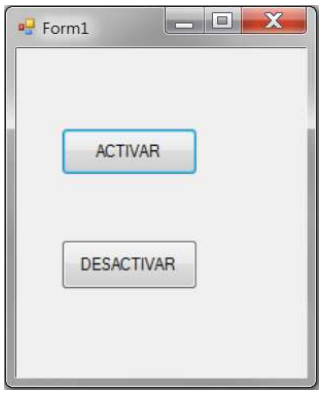
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using stx8xxx;
```

```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();

            PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Variable para retorno de estado del PLC.
            UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

            // Crear array de bytes a enviar (dos bytes).
            byte[] DataBytes = new byte[2];

            // Especificar el primer byte el valor "1", para activar DOUT1.
            DataBytes[0] = (byte)1;

            // Especificar el segundo byte un valor cualquiera.
            DataBytes[1] = (byte)55;

            // Enviar bytes UDP al script con el metodo "Send".
            PioBoard.Cmd.Udp.Send(DataBytes, 2, out OnUdpRxStat);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // Variable para retorno de estado del PLC.
            UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

            // Crear array de bytes a enviar (dos bytes).
            byte[] DataBytes = new byte[2];

            // Especificar el primer byte el valor "0", para desactivar DOUT1
            DataBytes[0] = (byte)0;

            // Especificar el segundo byte un valor cualquiera.
            DataBytes[1] = (byte)66;

            // Enviar bytes UDP al script con el metodo "Send".
            PioBoard.Cmd.Udp.Send(DataBytes, 2, out OnUdpRxStat);
        }
    }
}
```

Para generar una versión “Debug”, click en “Iniciar depuración”, del menú “Depurar”:



El programa se ejecutará mostrando la ventana principal:



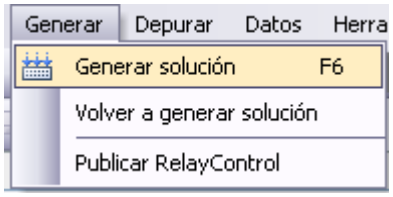
Al hacer click en el botón “ACTIVAR” se llamará al evento button1_Click(), el cual finalmente llamará al metodo PioBoard.Cmd.Udp.Send(), que será el responsable de enviar los bytes definidos en el array DataBytes[]. El script del PLC interpretará los bytes activando la salida DOUT1.

Análogamente, al hacer click en el “DESACTIVAR”, se llamara al evento button2_Click() que enviará bytes con el valor adecuado para desactivar a DOUT1.

Para que el ejemplo funcione es necesario cargar en el PLC un programa que procese los datos recibidos por la interfaz Ethernet / UDP. El programa del PLC se describe en la pagina 23.

Para detener o salir del programa, ciérrelo de la forma habitual, haciendo click en la X de la ventana.

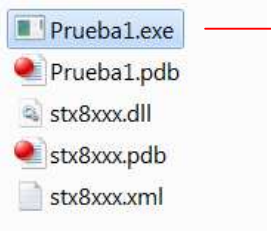
Para generar la versión “Release”, click en “Generar solución” de menú “Generar”:



El programa se compilara y creara el ejecutable **Prueba1.exe**. Busque el ejecutable en el directorio donde guardo la solución. Una vez dentro del directorio de la solución, en el siguiente directorio:

Prueba1\Prueba1\bin\Release

Se encontrará el ejecutable. Doble click en él y el programa se ejecutará nuevamente.



Ejecutable del programa Prueba1.

6.9 Cargar el Código de Prueba en PLC

El próximo paso es cargar un programa en el PLC, que nos permita recibir los bytes enviados desde la aplicaciones en C#.

Este script se explico en el Manual de Programación Pawn del PLC (**STX80XX-MP-PLC-AX_CX_DX**).

El siguiente programa activa el evento @OnUdpRx() para recibir datos a través de la interfaz Ethernet. Cuando un paquete UDP llegue a nuestro PLC se procesará el evento de la siguiente forma:

1. Obtener dirección IP del paquete recibido.
2. Leer 6 bytes del paquete recibido.
3. Comprobar valor del primer byte leído: Si es “1”, activar DOUT1. Si es “0”, desactivar DOUT1.

Mientras tanto, el programa principal conmutará el led del dispositivo cada 250 mS.

A continuación se muestra el código del programa en Pawn (también disponible para descargar de la Web):

```
// Crear las siguientes variables Globales a todo el script.

static RxData[6]         // Array para almacenar los datos UDP recibidos
static RxIP[4]           // Array para almacenar la IP de los datos recibidos
static RxPort  = 0       // Almacena el puerto UDP de los bytes recibidos.
```

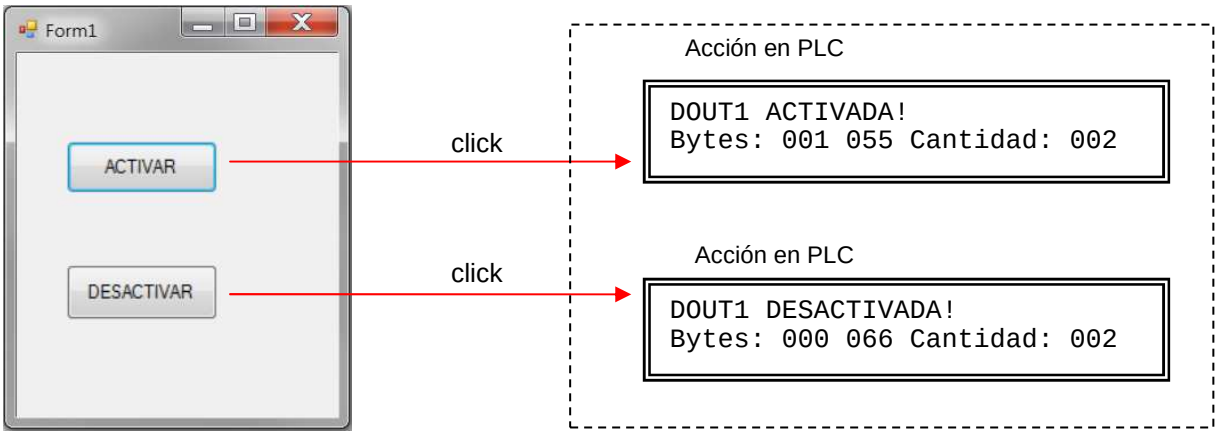
```
// ---------------------------------------------------------------------------
// FUNCIONES
// ---------------------------------------------------------------------------

// Función principal del programa.

PlcMain()
{
   // Vaciar Rx Buffer (Optional, recomendado como buena practica).
   UdpRxBufFlush()

   // Activar el evento para recibir datos UDP.
   UdpRxSetEvent()

   // Ciclo principal del programa.
   while(1)
   {
      // Conmutar led debug cada 250 mS.
      DelayMS(250)
      LedToggle()
   }
}

// Función del evento @OnUdpRx().

@OnUdpRx()
{
   // Obtener dirección IP remota y Puerto de los datos recibidos.
   // Nota: solo para fines didacticos.
   UdpRxGetAddr(RxIP[0], RxIP[1], RxIP[2], RxIP[3], RxPort)

   // Leer 6 bytes del buffer de recepcion.
   UdpRxDataRead(RxData, 0, 6, false)

   // Comprobar valor de primer byte recibido y activar/desactivar RELAY1/DOUT1.
   if(RxData[0] == 1)
      DoutSetOn(DOUT1)
   if(RxData[0] == 0)
      DoutSetOff(DOUT1)

   // Comprobar valor del segundo byte recibido y activar/desactivar RELAY2/DOUT2.
   if(RxData[1] == 1)
      DoutSetOn(DOUT2)
   if(RxData[1] == 0)
      DoutSetOff(DOUT2)

   // Comprobar valor del tercer byte recibido y activar/desactivar RELAY3/DOUT3.
   if(RxData[2] == 1)
      DoutSetOn(DOUT3)
   if(RxData[2] == 0)
      DoutSetOff(DOUT3)
}
```

6.10 Prueba de la Aplicación

Una vez cargado el programa Pawn en el PLC, procedemos a ejecutar la aplicación creada en Visual C#, llamada “Prueba1”, en nuestra computadora.

Verificamos antes si el PLC está conectado a nuestra computadora, mediante la red Ethernet.

A continuación, se muestra la ejecución del programa (izquierda), y sus efectos en el PLC (derecha) al hacer click en los botones:



Aplicación C#, Computadora
IP 192.168.1.11

Script PAWN, PLC
IP 192.168.1.81

En la ilustración superior, la computadora tiene asignada la dirección IP 192.168.1.11 y el PLC la dirección 192.168.1.81.

En el recuadro del PLC, se muestra los valores de los bytes y la cantidad que llegan al mismo.

Cuando se presiona el botón “ACTIVAR” llegan al PLC dos bytes con los valores “`001 055`”.

Significando, DataBytes[0] = 001 y DataBytes[1] = 055, tal como era de esperar. Recordemos que si el primer byte recibido tenía el valor “1”, la DOUT1 se debe activar. Si nos fijamos en el PLC la salida debe estar activada.

Si se presiona el botón “DESACTIVAR” llegan al PLC dos bytes con los valores “`001 066`”.

Significando, DataBytes[0] = 000 y DataBytes[1] = 066, tal como era de esperar. Recordemos que si el primer byte recibido tenía el valor “’0” la DOUT1 se debía desactivar. De nuevo, si nos fijamos en el dispositivo, la salida debe estar desactivada.

Ahora, usted ya sabe como enviar datos al dispositivo STX80XX en modo PLC y procesarlos… ¿ Simple verdad ?.

6.11 Comprobando Errores en la Aplicación

En este punto usted se pregunta: ¿ Qué pasa si al enviar un dato al PLC, ocurre un error, como puedo detectarlo ?.

La respuesta es simple, solo debe analizar las variables de estado, devueltas por las funciones de transmisión en C#. Por ejemplo, el método del evento click para el botón “ACTIVAR”, podría rescribirse como sigue:

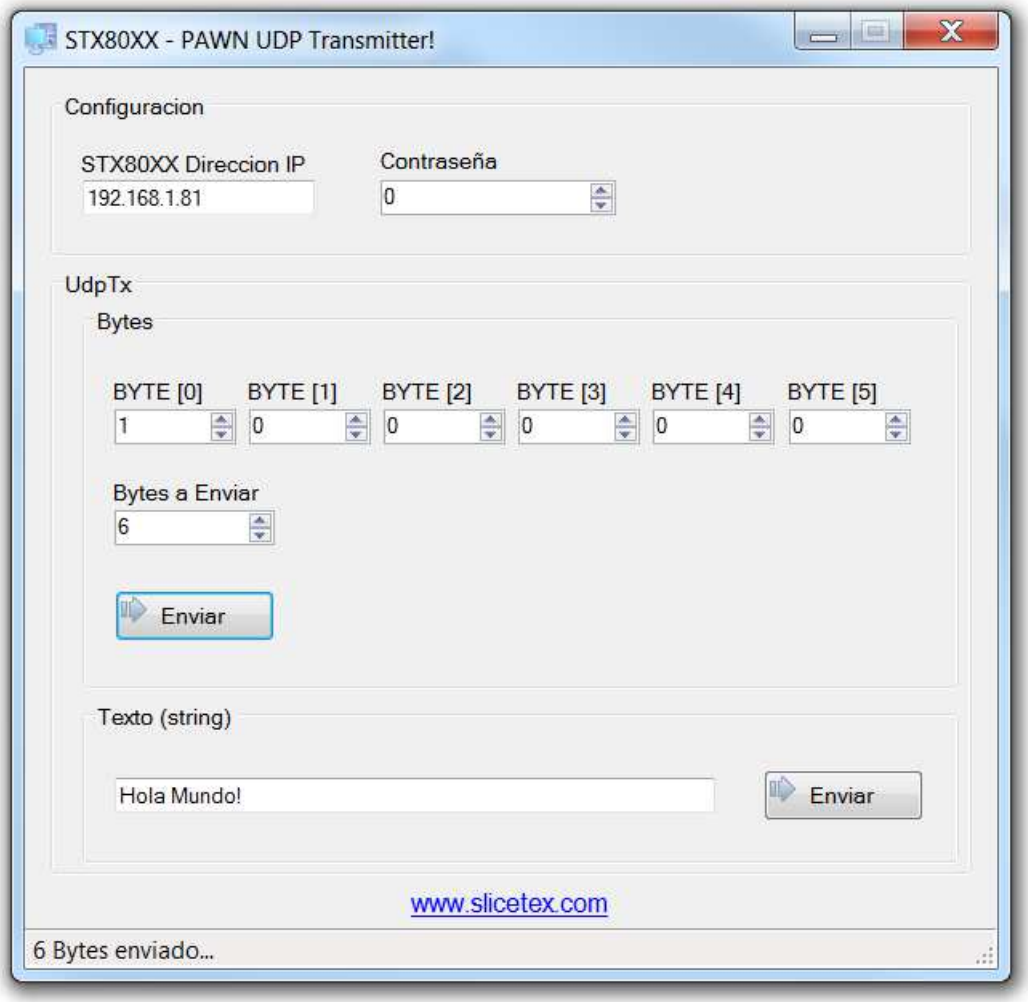
```
private void butSndBytes_Click(object sender, EventArgs e)
{
    // Variable para retorno de estado del comando enviado.
    SendStat CmdStat = SendStat.Success;

    // Variable para retorno de estado del PLC.
    UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

    // Crear array de bytes a enviar (dos bytes).
    byte[] DataBytes = new byte[2];

    // Especificar el primer byte el valor "1", para activar DOUT1.
    DataBytes[0] = (byte)1;

    // Especificar el segundo byte un valor cualquiera.
    DataBytes[1] = (byte)55;

    // Enviar bytes UDP al script con el metodo "Send", y almacenar retorno.
    CmdStat = PioBoard.Cmd.Udp.Send(DataBytes, 2, out OnUdpRxStat);

    // Send UDP bytes to script with command "UdpRx".
    CmdStat = PioBoard.Cmd.Udp.Send(DataBytes, DataSize, out OnUdpRxStat);

    // Comprobar respuesta de comando "UdpRx" (falla en conexion, etc).
    if (CmdStat != SendStat.Success)
    {
        MessageBox.Show("Codigo de Error:\n" + CmdStat.ToString(),
          "Error al enviar commando (UdpRx) ...",
          MessageBoxButtons.OK, MessageBoxIcon.Error);

        return;
    }

    // Comprobar respuesta del PLC al ejecutar la operacion "UdpRx" (buffer
    // del PLC lleno o no leido, demasiados bytes, etc.)
    if (OnUdpRxStat != UdpRxCmdStat.OK)
    {
        string ErrorCode = "Codigo de Error:\n" + OnUdpRxStat.ToString();

        MessageBox.Show(ErrorCode, "Error al ejecutar operacion (UdpRx) ...",
          MessageBoxButtons.OK, MessageBoxIcon.Error);

        return;
    }
}
```

En el código anterior se comprueba el retorno del comando en almacenado en la variable “CmdStat” del tipo `SendStat`, y el retorno de la operación en OnUdpRxStat. Si existe un error, se mostrara un cartel (MessageBox) indicando el tipo de error. En la pagina 41, se explica en detalle los valores devueltos.

6.12 Mas Ejemplos

Al instalar el STX80XX-SDK (Software Development Kit) se instalan ejemplos de programación en C#. El programa ***PawnUdpTx*** puede serle útil para enviar datos Ethernet / UDP al PLC.

Su código fuente esta en el directorio "**visual_cs\interface\PawnUdpTx**".



***PawnUdpTx**: ventana principal.*

7 Diseño de Aplicación para Recibir Datos

En esta sección mostraremos un procedimiento practico para crear un programa que recibe datos en C# desde nuestro PLC.

El programa escuchara datos Ethernet / UDP provenientes del PLC, en el puerto 4980. Cada byte que llegue será convertido en un carácter ASCII para visualizarlo en pantalla.

Se diseñará un programa Pawn para el PLC que enviará constantemente los caracteres “Hola!”, repitiéndolos cada 1 segundo. Los caracteres serán enviados desde el PLC a la IP de la computadora que ejecuta la aplicación C#, en el puerto 4980.

Los pasos para crear un programa en C#, se explico con detalle en la sección Diseño de Aplicación para Transmitir Datos, pagina 5. Es muy importante, que primero entienda el procedimiento de diseño, antes de continuar.

El programa en C#, se diseñara con un botón llamado “ESCUCHAR”, que al hacer click sobre él, intentara leer datos que lleguen vía Ethernet del PLC. Si llegan los datos, los mostrara en un cartel o MessageBox.

Adjunto a este documento, podrá encontrar el programa completo, listo para compilar y ejecutar en Microsoft Visual C# 2005. El proyecto se llama “**Prueba2**”.

Nota: Ejemplo disponible para descargar en Microsoft Visual Basic .Net también.

7.1 Diseño de Programa en C#

En Microsoft Visual C#, desde la ventana de diseño, agregamos un botón al formulario, y lo nombramos “ESCUCHAR”, como se muestra a continuación:

Luego hacemos doble click en el botón “ESCUCHAR” de formulario, para agregar el siguiente código que manejara el evento “click” en el formulario:

```
private void button1_Click(object sender, EventArgs e)
{
    byte[] Packet;
    string DataStr = "empty";

    // Esperar 5 segundos por un paquete UDP.
    // Comprobar que fue recibido.
    if(PioBoard.Cmd.Udp.Receive(4980, 5, out Packet)==UdpReceiveStat.Success)
    {
        // PAQUETE RECIBIDO!!!

        // CONVERTIR PAQUETE RECIBIDO EN CARACTERES (STRING)

        for (int i = 0; i < Packet.Length; i++)
        {
            if (i == 0)
            {
                // Primer Byte.
                DataStr = string.Format("{0},", (char)Packet[i]);
            }
            else
            {
                // Resto de los Bytes.
                DataStr += string.Format("{0},", (char)Packet[i]);
            }
        }

        // MOSTRAR PAQUETE RECIBIDO EN PANTALLA.
        MessageBox.Show(DataStr, "Paquete Recibido!", MessageBoxButtons.OK,
           MessageBoxIcon.Information);
    }
    else
    {
        // NO SE RECIBIO NINGUN PAQUETE.
        MessageBox.Show("Intente Nuevamente...", "No se Recibieron Datos",
           MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Al hacer click en el botón “ESCUCHAR”, se llamará al método `button1_Click()` que maneja el evento click. Desde el evento click, llamamos a la función para escuchar paquetes UDP del PLC: `PioBoard.Cmd.Udp.Receive()`.

El primer argumento de la función es el puerto a escuchar, 4980. El segundo argumento, es la cantidad de tiempo en segundos a esperar por un paquete, en este caso utilizamos 5 segundos de Timeout. El tercer argumento, es el array de bytes donde los datos recibidos serán almacenados, en este caso utilizamos el array llamdo “Packet”.

El método Receive() también puede retornar IP y Puerto remoto del dispositivo que envió los datos, vea pagina 46 para mas info.

Al llamar a la método Receive(), se comprueba su retorno, si es `UdpReceiveStat.Success`, significa que llegaron nuevos datos. Más información en página 47, Enumeración UdpReceiveStat.

Si el paquete fue recibido, los bytes en Packet[] son convertidos a caracteres y almacenados en una cadena, en el loop:

```
// CONVERTIR PAQUETE RECIBIDO EN CARACTERES (STRING)

for (int i = 0; i < Packet.Length; i++)
{
    if (i == 0)
    {
        // Primer Byte.
        DataStr = string.Format("{0},", (char)Packet[i]);
    }
    else
    {
        // Resto de los Bytes.
        DataStr += string.Format("{0},", (char)Packet[i]);
    }
}
```

Finalmente, la cadena resultante se imprime en pantalla a través de una MessageBox:

```
// MOSTRAR PAQUETE RECIBIDO EN PANTALLA.
MessageBox.Show(DataStr, "Paquete Recibido!", MessageBoxButtons.OK,
   MessageBoxIcon.Information);
```

Si el paquete no se recibe dentro de los 5 segundos, el método Receive() no retorna: `UdpReceiveStat.Success`, y se ejecuta el código en el "**else**":

```
else
{
    // NO SE RECIBIO NINGUN PAQUETE.
    MessageBox.Show("Intente Nuevamente...", "No se Recibieron Datos",
       MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Dicho código, mostrara en una MessageBox que no llegaron datos.

7.2 Cargar Codigo de Prueba en el PLC

El próximo paso es cargar un programa en el PLC, que nos permita enviar datos a la aplicación en C#.

Este programa se explicó en el Manual de Programación Pawn del PLC (**STX80XX-MP-PLC-AX_CX_DX**).

El siguiente programa en Pawn, envía 5 bytes a la dirección IP 192.168.1.11, puerto UDP numero 4980. Los primeros 5 bytes contienen los siguientes caracteres ASCII: 'H', 'o', 'l', 'a', '!'.

Modifique la IP de destino, por la que corresponde a su computadora.

Los bytes se envían cada 1 segundo.

Ver código en página siguiente.

```
PlcMain()
{
   // Crear un array para almacenar los bytes a enviar.
   new Data[5]

   // Almacenar caracteres ASCII a enviar...
   Data[0] = 'H'
   Data[1] = 'o'
   Data[2] = 'l'
   Data[3] = 'a'
   Data[4] = '!'

   // Ciclo principal del programa.
   while(1)
   {
      // Enviar paquete UDP a dirección 192.168.1.15, puerto 4980.
      UdpSend(192,168,1,15, 4980, 5, Data, false)

      // Pausar programa por 1 segundo y conmutar led debug.
      DelayS(1)
      LedToggle()
   }
}
```

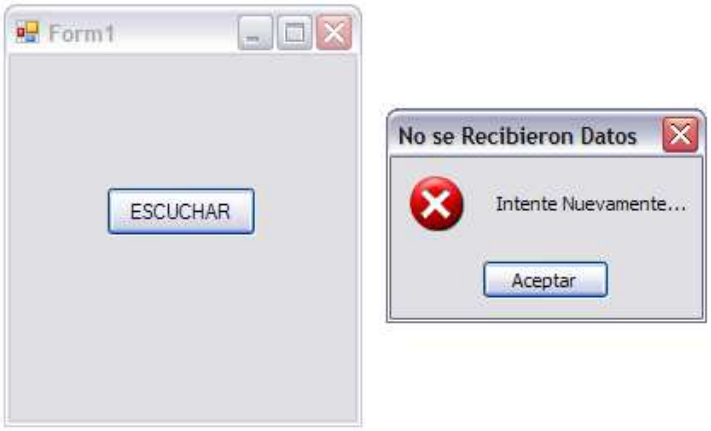
El programa está disponible para descargar desde nuestra página Web junto con esta nota de aplicación.

7.3 Prueba de la Aplicación

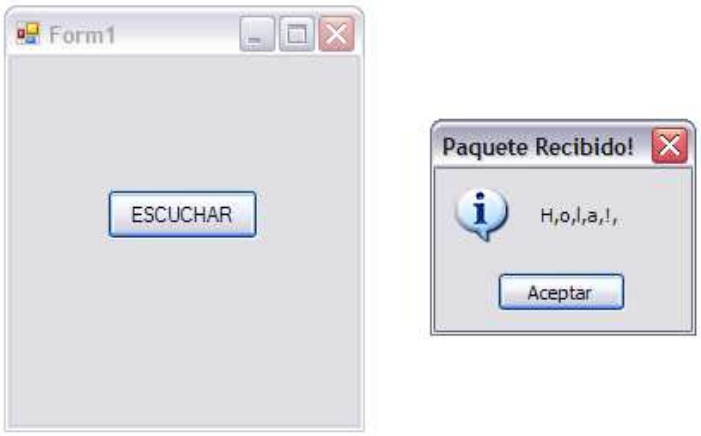
Una vez cargado el programa en el PLC, procedemos a ejecutar la aplicación creada en Visual C#, llamada “Prueba2”, en nuestra computadora.

Verificamos antes si el PLC está conectado correctamente a nuestra computadora, mediante la red Ethernet.

A continuación, se muestra la ejecución del programa. Si presionamos el botón “ESCUCHAR” y los datos no son recibidos dentro de los 5 segundos, el método Receive() agota su espera (Timeout) y el programa muestra el siguiente cartel de error (“No se Recibieron Datos”):



Si los datos, se reciben dentro de los 5 segundos, el método Receive() retorna los bytes recibidos, y el programa los imprime en pantalla como se muestra a continuación:



Los caracteres se imprimen en el cartel “Paquete Recibido!”, separados cada uno por una “,”, de la siguiente forma: “H,o,l,a,!,”.

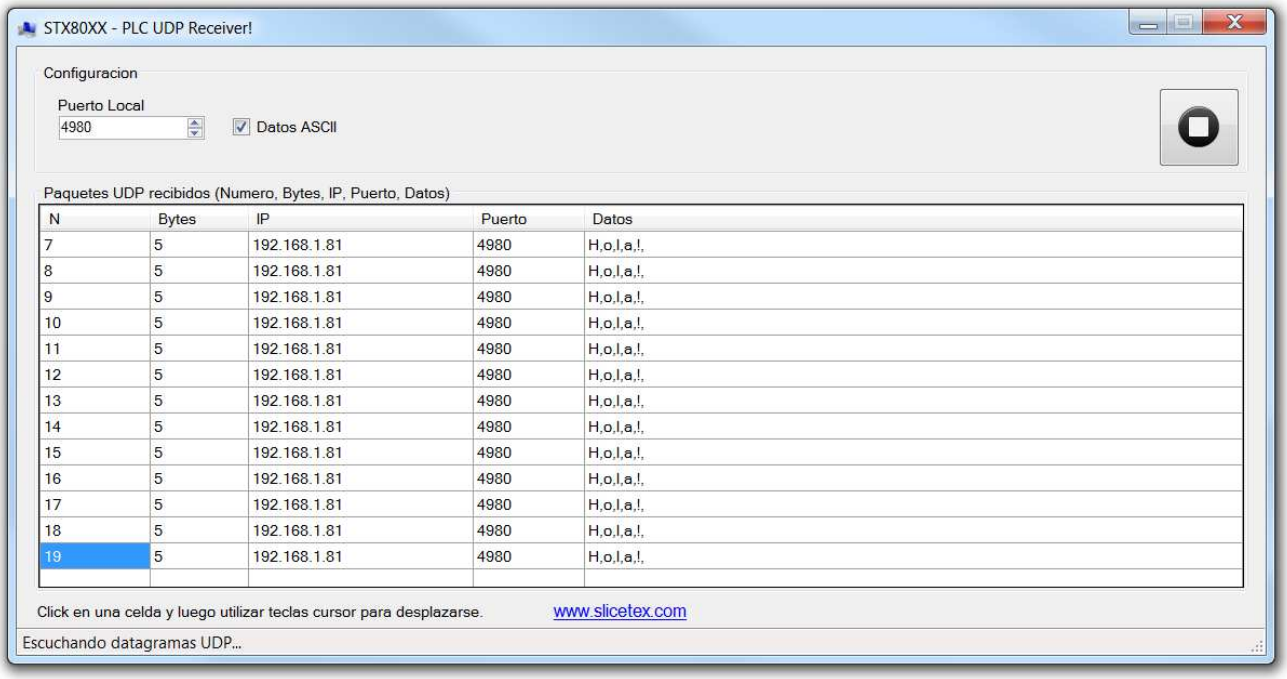
7.4 Mas Ejemplos

Al instalar el STX80XX-SDK (Software Development Kit) se instalan ejemplos de programación en C#. El programa ***PawnUdpRx*** puede serle útil para recibir datos Ethernet / UDP desde el PLC.

Su código fuente esta en el directorio “visual_cs\interface\PawnUdpRx”.

En el programa ***PawnUdpRx*** se ejemplifica como obtener la IP y Puerto remoto de los datos recibidos. Además, se muestra como “escuchar” los datos en segundo plano, utilizando un “Background Worker” de C#. Concepto muy útil para comunicaciones. **Se recomienda estudiar este ejemplo.**

*“El control **BackgroundWorker** fue introducido en el .NET Framework 2.0 y permite realizar operaciones costosas o duraderas en **un hilo diferente al del interfaz**, de manera que la aplicación sigue respondiendo al usuario mientras este trabajo se procesa en **segundo plano**”.*



***PawnUdpRx**: Recibiendo datos UDP vía Ethernet del PLC.*

8 Librería STX8XXX.DLL de C#

En esta sección describiremos la librería STX8XXX.DLL, necesaria para enviar y recibir datos Ethernet / UDP al PLC.

La descripción de la librería es superficial, centrándose en los métodos, clases, objetos y constantes que se utilizan para el Modo PLC del dispositivo STX80XX.

Para una descripción total de la librería, recomendamos leer el "Manual de Usuario Modo DAQ", **STX80XX-UM-DAQ-AX_BX**. En aquel documento, se listan todas las capacidades de la librería, pero tenga en cuenta que muchos métodos, son para utilizar en el modo "DAQ" del dispositivo STX80XX.

8.1 Organización

La librería consiste en un set de objetos o clases accesibles desde la clase principal **Stx8xxx**.

La clase **Stx8xxx** contiene métodos, constantes, estructuras y objetos, que son útiles para enviar comandos y recibir datos desde el dispositivo STX80XX. En el documento, cuando "instanciemos" la clase **Stx8xxx**, le daremos el nombre de **PioBoard**.

Instanciar una clase u objeto, significa crearlo en memoria, por lo general con la palabra "new".

8.2 Como leer los Métodos

En este documento, los métodos se presentan en el siguiente formato:

Método(Arg1, Arg2): Método de prueba.		
Argumentos	**Tipo**	**Descripción**
Arg1	E	Argumento de entrada.
Arg2	S	Argumento de Salida. Se retorna un valor en la variable Arg2.
Retorno	**Tipo**	**Descripción**
Tipo	S	Operación exitosa.

Donde:

- **Método(Arg1, Arg2)**: Prototipo del método, que incluye nombre de argumentos.
- **Arg1 y Arg2**: Argumentos o parámetros del método. Siempre, observar la naturaleza del argumento (string, entero, clase, etc.).
- **Tipo**: Si es "E" significa entrada. "S" significa salida (en caso de ser un argumento, se entiende que el método retornara un valor en la variable). Si es "E/S" el argumento es de entrada y salida.
- **Retorno**: El retorno del método, es su resultado, puede ser un constante, en ese caso se listan todas las constantes posibles o se especifica el nombre de la enumeración, de lo contrario se le da nombre al retorno con fines descriptivos.

9 Clase Principal Stx8xxx

9.1 Objetivo

Proveer acceso a estructuras de datos, constantes, objetos y métodos, que permiten enviar comandos y recibir datos desde el dispositivo STX80XX.

Mantener datos en variables para configurar la librería. Inicializa y configura inicialmente la librería STX8XXX.

9.2 Constructores

En el documento, cuando "instanciemos" la clase **Stx8xxx**, le daremos el nombre de **PioBoard**.

Stx8xxx(): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería [1].		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Por defecto: IP=192.168.1.81, Port=4950, Password=0, Timeout=2, Retries=1, dispositivo target = STX8081.

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx();
```

Stx8xxx(string IP, Stx8xxxId TargetDevice): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería [1].		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Por defecto: Port=4950, Password=0, Timeout=2, Retries=1.

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", Stx8xxxId.STX8081);
```

Stx8xxx(string IP, UInt32 Password, Stx8xxxId TargetDevice): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería .		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Password	E	Clave o password para utilizar en el dispositivo.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
-		Por defecto: Port=4950, Timeout=2, Retries=1

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
```

Stx8xxx(string IP, UInt32 Password, int Port, int Timeout, int Retries, Stx8xxxId TargetDevice): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería .		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Password	E	Clave o password para utilizar en el dispositivo.
Port	E	Puerto del CSP (Command Server Protocol). Ej: 4950.
Timeout	E	Tiempo a esperar por la respuesta de un comando (en segundos).
Retries	E	Números de intentos de envío de un comando cuando ocurre un Timeout.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
-		

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0,
                       (int)Stx8xxxUdpPorts.CmdServer,
                       2, 1, Stx8xxxId.STX8081);
```

Stx8xxx(string IP_Or_Hostname, UInt32 Password, Stx8xxxId TargetDevice, Stx8xxxAddressTypeFormat Type): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería.		
Argumentos	**Tipo**	**Descripción**
IP_Or_Hostname	E	Dirección IP o hostname del dispositivo STX80XX.
Password	E	Clave o password para utilizar en el dispositivo.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Type	E	Tipo de dirección a utilizar: Hostname = Nombre de host. IP = Dirección IP. Unknown= Auto detectar.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Este método permite utilizar un nombre en vez de una dirección IP.

Ejemplo:

El siguiente código inicializa la librería para utilizar un dispositivo conectado a la dirección “myhome.net” en vez a una dirección IP estática como “192.168.1.81”.

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("myhome.net", 0,
                       Stx8xxxId.STX8081,
                       Stx8xxxAddressTypeFormat.Hostname);
```

Notas:

Stx8xxxUdpPorts.CmdServer: Contiene el numero de puerto UDP del Command Server Protocol.

9.3 Objetos Miembros

La clase contiene dos objetos:

1. **Cmd** : Contiene métodos y objetos para enviar comandos al dispositivo y otras utilidades.
2. **Globals**: Provee información global, métodos de configuración y métodos varios de la librería.

9.4 Objeto PioBoard.Cmd

Contiene objetos que comandan el dispositivo. Comúnmente se los emplea para utilizar las características disponibles del dispositivo.

Objetos disponibles en "Cmd"	
Objeto	**Descripción**
Board	Métodos varios para controlar el dispositivo STX80XX en general.
BoardConfig	Configuración del dispositivo.
BoardInfo	Obtiene información del dispositivo.
Count	Control de entradas de contadores (COUNTx).
Din	Control de entradas digitales o discretas (DINx).
Dout	Control de salidas digitales o discretas (DOUTx)
Lcd	Control de display LCD.
NetHmi	Permite enviar y recibir paquetes NetHMI del dispositivo en modo PLC.
PWM	Control de salidas PWM (PWMx).
Relay	Control de salidas RELAY (RELAYx).
Udp	Interface para enviar datos UDP del dispositivo en modo PLC.
Vin	Control de entradas analógicas (VINx).
Vout	Control de salida analógica (VOUTx).

El acceso a los objetos en "Cmd", se explica con detalles en el documento **STX80XX-UM-DAQ-AX_BX**. Desde "Cmd" es posible enviar los comandos al dispositivo STX80XX.

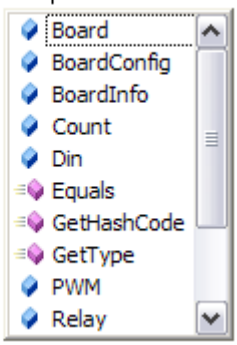
10 Objetos para Enviar Comandos

Esta sección se describirá como enviar los comandos que permiten utilizar las características del dispositivo STX80XX.

Los ejemplos suponen que usted ya inicializo la librería STX8XXX.DLL correctamente, si no lo hizo, lea las secciones anteriores (por ejemplo, diríjase a la página 5).

Los objetos para enviar los comandos, son accesibles desde el objeto “Cmd”:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx();

// Se accede a los objetos de “Cmd”:
PioBoard.Cmd.
```




```
// Enviar bytes UDP al PLC con el objeto “Udp”.
PioBoard.Cmd.Udp.Send(DataBytes, 2, out OnUdpRxStat);
```

Antes de continuar, explicaremos algunas estructuras y enumeraciones útiles, que son utilizadas con frecuencia al momento de enviar comandos al dispositivo STX80XX.

10.1 Enumeración SendStat

La enumeración "**SendStat**" contiene constantes que son devueltas por la mayoría de los comandos que se utilizan para comandar el dispositivo STX80XX. Hay algunas excepciones, que se comentarán en su debida sección. Se suele denominar al proceso de enviar comando y recibir una respuesta, como "**transacción**".

Estas constantes permiten conocer el estado de retorno al enviar y ejecutar un comando en el dispositivo STX80XX. Un retorno exitoso, comando ejecutado correctamente, debería devolver ***SendStat.Success***, de lo contrario, si no se devuelve ese valor, existió algún error.

A continuación se lista la definición de la enumeración "**SendStat**" con sus posibles valores:

enum SendStat : int: Enumeracion que representa el estado de la transaccion de un comando.			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
Success	int	0	Retorno exitoso de función.
ErrorTimeout	int	-1	Tiempo de espera por respuesta de comando expiro.
ErrorUnknown	int	-2	Otro error, mirar código fuente de librería.
ErrorIncompleteHeaders	int	-3	Paquete incompleto (headers/cabeceras no recibidas).
ErrorIncompleteOutput	int	-4	Headers recibidos pero recepción de datos de salida de comando incompletos.
ErrorSend	int	-5	Se ha retornado un error cuando se intento acceder al socket para transmitir datos.
ErrorSendIncomplete	int	-6	Transmisión incompleta.
ErrorCmdRtnStat	int	-7	CRP (Command Return Packet) fue recibido, pero el campo STAT0 contiene un código de error (error en la ejecución de comando). Comprobar retorno de comando, campos STAT0 y STAT1 por mas información.
ErrorRtnProtocolVersion	int	-8	Versión de protocolo CSP retornado desde la STX80XX no está soportada por esta librería.
ErrorProtocolVerUnsupported	int	-9	Versión de protocolo CSP utilizado por esta librería, no está soportada por la STX80XX.
ErrorPasswordIncorrect	int	-10	El comando no fue ejecutado debido a que el password es incorrecto. Comprobar password!.
ErrorTypeNotExist	int	-11	El tipo de comando para ejecutar en la STX80XX no existe. Comprobar campo TYPE.
ErrorCmdUnknown	int	-12	El comando que se está tratando de ejecutar no existe. Comprobar el campo ID.

enum SendStat : int: Enumeracion (continuacion).			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
ErrorCmdReturnError	int	-13	El comando ejecutado ha retornado un error. Quizás el tipo de error fue retornado en la salida de datos del comando (bytes del campo R[0]…R[N]). Mire la descripción del comando para más información.
ErrorRxUnknown	int	-14	Error desconocido en la recepción del paquete. Mirar código fuente.
ErrorIncompleteArgs	int	-15	Argumentos del comando enviado a la STX80XX están incompletos. Comprobar valor de campo ARGSIZE y campos de datos ARGS.
ErrorWrongArguments	int	-16	Los argumentos del comando enviado son incorrectos
ErrorCmdUnsopportedInPLC	int	-17	El comando que está tratando de ejecutar no está soportado cuando el modo PLC está activo en el dispositivo. Comprobar que el ID del comando sea un valor entre 0 y 19.
ErrorCmdUnsopportedInBootloader	int	-18	El comando que está tratando de ejecutar no está soportado cuando el modo Bootloader está activo en el dispositivo. Comprobar el ID del comando o cambiar el modo de funcionamiento del dispositivo a PLC o DAQ.
ErrorOnSocketBindOrIsBusy	int	-19	Otro recurso está intentando acceder al socket de conexión al mismo tiempo.
ErrorResolvHostnameIP	int	-20	No se puede obtener la dirección IP del Hostname. Comprobar conexión internet, nombre de Hostname valido y configuración DNS.

Valores de retorno distinto a “0” o “Success” desde el dispositivo significan error.

10.2 Objeto PioBoard.Cmd.Udp

Envía paquetes UDP a un dispositivo en modo PLC y contiene métodos para recibir datos UDP.

10.2.1 Métodos

SendStat Send(byte[] Data, out UdpRxCmdStat RxStat): Envia bytes UDP al dispositivo en modo PLC.		
Argumentos	**Tipo**	**Descripción**
Data	E	Array de bytes a enviar.
RxStat	S	Código de estado al ejecutar el comando “UdpRx” el PLC. Ver enumeración UdpRxCmdStat, pag. 47.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat, pag. 41.
Notas		**Descripción**
1		Recuerde especificar correctamente IP y contraseña del dispositivo en la librería, antes de utilizar este método.

Ejemplo:

```
// Variable para retorno de estado del PLC.
UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

// Crear array de bytes a enviar (dos bytes).
byte[] DataBytes = new byte[2];

// Especificar el primer byte el valor "1".
DataBytes[0] = (byte)1;

// Especificar el segundo byte el valor 55.
DataBytes[1] = (byte)55;

// Enviar bytes UDP al script con el metodo "Send".
PioBoard.Cmd.Udp.Send(DataBytes, out OnUdpRxStat);
```

SendStat Send(byte[] Data, byte DataSize, out UdpRxCmdStat RxStat): Envia bytes UDP al dispositivo en modo PLC.		
Argumentos	**Tipo**	**Descripción**
Data	E	Array de bytes a enviar.
DataSize	E	Numero de bytes a enviar del array “Data”.
RxStat	S	Código de estado al ejecutar el comando “UdpRx” el PLC. Ver enumeración UdpRxCmdStat, pag. 47.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat, pag. 41.
Notas		**Descripción**
1		Recuerde especificar correctamente IP y contraseña del dispositivo en la librería, antes de utilizar este método.

Ejemplo:

```
// Variable para retorno de estado del PLC.
UdpRxCmdStat OnUdpRxStat = UdpRxCmdStat.OK;

// Crear array de bytes a enviar (dos bytes).
byte[] DataBytes = new byte[2];

// Especificar el primer byte el valor "1".
DataBytes[0] = (byte)1;

// Especificar el segundo byte el valor 55.
DataBytes[1] = (byte)55;

// Enviar solo un byte al script con el metodo "Send".
PioBoard.Cmd.Udp.Send(DataBytes, 1, out OnUdpRxStat);
```

UdpReceiveStat Receive(int Port, double Timeout, out byte[] Packet): Escucha el puerto “Port”, esperado un paquete UDP. La funcion retorna luego de un tiempo especificado.		
Argumentos	**Tipo**	**Descripción**
Port	E	Puerto local UDP a escuchar.
Timeout	E	Tiempo a esperar un paquete UDP. Se especifica en segundos.
Packet	S	Array de retorno con los bytes de datos del paquete UDP recibido.
Retorno	**Tipo**	**Descripción**
UdpReceiveStat	S	Estado de operación. Ver enumeración UdpReceiveStat, pag. 47.
Notas		**Descripción**
-		

Ejemplo:

```
byte[] Packet;

// Esperar 0.5 segundos por un paquete UDP, en puerto 4908.
// Comprobar que fue recibido.

if(PioBoard.Cmd.Udp.Receive(4980,0.5,out Packet)==UdpReceiveStat.Success)
{
    // PAQUETE RECIBIDO!!!
}
else
{
    // NO SE RECIBIO NINGUN PAQUETE.
}
```

UdpReceiveStat Receive(int Port, double Timeout, out string RemoteIP, out int RemotePort, out byte[] Packet): Escucha el puerto “Port”, esperado un paquete UDP. La funcion retorna luego de un tiempo especificado.

Argumentos	**Tipo**	**Descripción**
Port	E	Puerto local UDP a escuchar.
Timeout	E	Tiempo a esperar un paquete UDP. Se especifica en segundos.
RemoteIP	S	Retorno con Dirección IP remota del paquete recibido.
RemotePort	S	Retorno con Puerto UDP remoto del paquete recibido.
Packet	S	Array de retorno con los bytes de datos del paquete UDP recibido.
Retorno	**Tipo**	**Descripción**
UdpReceiveStat	S	Estado de operación. Ver enumeración UdpReceiveStat, pag. 47.
Notas		**Descripción**
-		

Ejemplo:

```
byte[] Packet;
string RemoteIP=null;
int RemotePort;

// Esperar 5 segundos por un paquete UDP, en puerto 4908.
// Almacenar IP y Puerto remoto.
// Comprobar que fue recibido.

if(PioBoard.Cmd.Udp.Receive((int) numLocalPort.Value, 5,
   out RemoteIP, out RemotePort, out Packet)==UdpReceiveStat.Success)
{
    // PAQUETE RECIBIDO!!!
}
else
{
    // NO SE RECIBIO NINGUN PAQUETE.
}
```

10.2.2 Enumeración UdpRxCmdStat

UdpRxCmdStat : byte : Constantes con codigo de retorno de un comando UdpRx.			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
OK	byte	0	Operación Exitosa.
ERROR_DISABLED	byte	1	Evento @OnUdpRx() en PLC desactivado.
ERROR_INVALID_DATA_SIZE	byte	2	El número de bytes enviados es invalido.
ERROR_OVERRUN	byte	3	Buffer de recepción (Rx Buff) contiene datos que no han sido leídos todavía por el script del PLC. El paquete enviado se descarto. Espere hasta que el script del PLC lea su buffer de recepción.
ERROR_CMD_FAIL	byte	200	Falla interna en comando “UdpRx”.

10.2.3 Enumeración UdpReceiveStat

UdpReceiveStat : int : Constantes con codigo de retorno de un comando UdpRx.			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
Success	int	0	Operación Exitosa.
ErrorTimeout	int	-1	Tiempo de espera agotado.
ErrorUnknown	int	-2	Error indeterminado.
ErrorSocketException	int	-3	Error de Excepción del Socket.

11 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **DAQ**: Data Aquisition (Adquisición de Datos).
- **Modo** PLC: Permite programar un dispositivo STX80XX mediante lenguajes Pawn/Ladder y luego ejecutarlos autónomamente para realizar algún tipo de control.
- **Modo DAQ**: Permite controlar al dispositivo STX80XX a través de una computadora conectada a la interfaz Ethernet, ya sea para adquirir datos o controlar las salidas del dispositivo.
- **UDP**: User Datagram Protocol. Protocolo orientado a la transmisión/recepción de datos. En el dispositivo STX80XX se usa para intercambiar datos mediante la interfaz Ethernet.
- **IP**: Dirección Internet, conformada por cuatro octetos, por ejemplo 192.168.1.81.
- **Ethernet**: Red de computadoras, que generalmente se utilizan el protocolo de internet TCP/IP o UDP/IP.

12 Historial de Revisiones

Tabla: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
04 03/ABR/2015	1	1. Ejemplos en Visual Basic .Net disponibles para nota de aplicación.	Preliminar
03 08/MAR/2015	1	1. Documentación adaptada a la línea STX80XX.	Preliminar
02 09/SEP/2012	1	1. Documentación adaptada al entorno StxLadder.	Preliminar
01 03/SEP/2010	1	1. Versión preliminar liberada.	Preliminar

13 Referencias

Ninguna.

14 Información Legal

14.1 Aviso de exención de responsabilidad

General: La información de este documento se da en buena fe, y se considera precisa y confiable. Sin embargo, Slicetex Electronics no da ninguna representación ni garantía, expresa o implícita, en cuanto a la exactitud o integridad de dicha información y no tendrá ninguna responsabilidad por las consecuencias del uso de la información proporcionada.

El derecho a realizar cambios: Slicetex Electronics se reserva el derecho de hacer cambios en la información publicada en este documento, incluyendo, especificaciones y descripciones de los productos, en cualquier momento y sin previo aviso. Este documento anula y sustituye toda la información proporcionada con anterioridad a la publicación de este documento.

Idoneidad para el uso: Los productos de Slicetex Electronics no están diseñados, autorizados o garantizados para su uso en aeronaves, área médica, entorno militar, entorno espacial o equipo de apoyo de vida, ni en las aplicaciones donde el fallo o mal funcionamiento de un producto de Slicetex Electronics pueda resultar en lesiones personales, muerte o daños materiales o ambientales graves. Slicetex Electronics no acepta ninguna responsabilidad por la inclusión y / o el uso de productos de Slicetex Electronics en tales equipos o aplicaciones (mencionados con anterioridad) y por lo tanto dicha inclusión y / o uso es exclusiva responsabilidad del cliente.

Aplicaciones: Las aplicaciones que aquí se describen o por cualquiera de estos productos son para fines ilustrativos. Slicetex Electronics no ofrece representación o garantía de que dichas aplicaciones serán adecuadas para el uso especificado, sin haber realizado más pruebas o modificaciones.

Los valores límites o máximos: Estrés por encima de uno o más valores límites (como se define en los valores absolutos máximos de la norma IEC 60134) puede causar daño permanente al dispositivo. Los valores límite son calificaciones de estrés solamente y el funcionamiento del dispositivo en esta o cualquier otra condición por encima de las indicadas en las secciones de Características de este documento, no está previsto ni garantizado. La exposición a los valores limitantes por períodos prolongados puede afectar la fiabilidad del dispositivo.

Documento: Prohibida la modificación de este documento en cualquier medio electrónico o impreso, sin autorización previa de Slicetex Electronics por escrito.

15 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

Para consultas y ventas envíe un mail a: info@slicetex.com

Para soporte técnico ingrese a nuestro foro en: www.slicetex.com/foro

Ing. Boris Estudiez

Slicetex Electronics
Córdoba, Argentina

16 Contenido

www.slicetex.com