ESCUELA SUPERIOR DE INFORMÁTICA

UNIVERSIDAD DE CASILLA-LA MANCHA

Automatización Industrial

Programación de la Interfaz FTI con C#

Tercera práctica

Jose Domingo López López
josed.lopez1@alu.uclm.es

Raúl Arias García
raul.arias2@alu.uclm.es

Grupo: ELF 08 (Tarde)
26 de Marzo del 2009

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

La programación *on-line* permite a los usuarios la creación y ejecución del código en el PC. Este modo nos posibilita una gran versatilidad, puesto que no dependemos de la capacidad de la interfaz si no que dependemos del PC en cuestión. Esto nos permite la generación de código más complejo y nos da la libertad de utilizar cualquier lenguaje de programación, entorno de programación y no estamos limitados por el tamaño del código.

2. Entorno de Programación

El lenguaje de programación que utilizaremos para realizar las prácticas será **C#.**

C# es un lenguaje de programación orientado a objetos y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. Sus sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar a JAVA aunque incluye mejoras derivadas de otros lenguajes.

Nos hemos decidido por éste dada su flexibilidad para realizar interfaces gráficas y por aprender un nuevo lenguaje, ya que la plataforma .NET es muy demandada en el mundo laboral hoy en día. Como entorno de programación utilizaremos **Microsoft Visual Studio 2008 C#** aunque también tenemos alternativas gratis y libres **#develop (Sharp Develop)** y la **plataforma Mono**.

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Además permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión 6). Así se pueden

crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

3. Programación de la interfaz FTI

Inicialmente disponemos de la librería *hckFish* que está escrita en lenguaje C, por lo que necesitaremos una adaptación a C#. Esta adaptación, que recibe el nombre de **FishFa30** y ha sido realizada por Ulrich-Müller, puede ser descargada de su página Web (ver enlace 1).

A continuación se muestra cómo crear un proyecto en Microsoft Visual Studio C#, un resumen de la librería comentando los métodos que utilizaremos, y dos ejemplos en los que se ilustra cómo manejar las entradas y salidas digitales.

3.1. Cómo crear un proyecto en Microsoft Visual Studio C#

1. El primer paso consiste en incluir el fichero *umFish30.dll* en el directorio C:\Windows\system32.
2. A continuación instalaremos el Microsoft Visual Studio C# (en este documento no se hará una descripción detallada de este paso).
3. Una vez instalado, al ejecutar la aplicación se nos mostrará la pantalla principal (ver Ilustración 1).

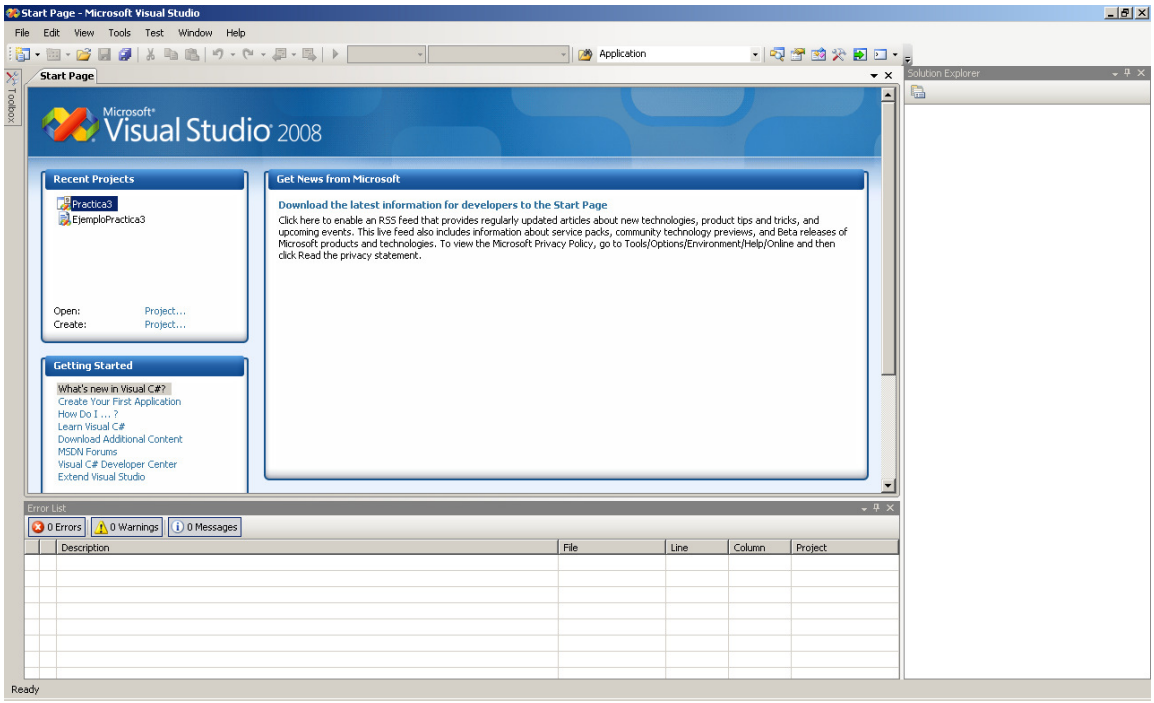

Ilustración 1: *Pantalla de inicio de Microsoft Visual Studio 2008 C#*

4. Navegamos en el menú de herramientas a través de la ruta *Archivo* > *Nuevo* > *Proyecto…* y seleccionamos la opción *Aplicación de consola* para crear un programa con interfaz basada en texto. A continuación elegimos el nombre de nuestro proyecto y decidimos en qué directorio deseamos guardarlo (ver Ilustración 2).

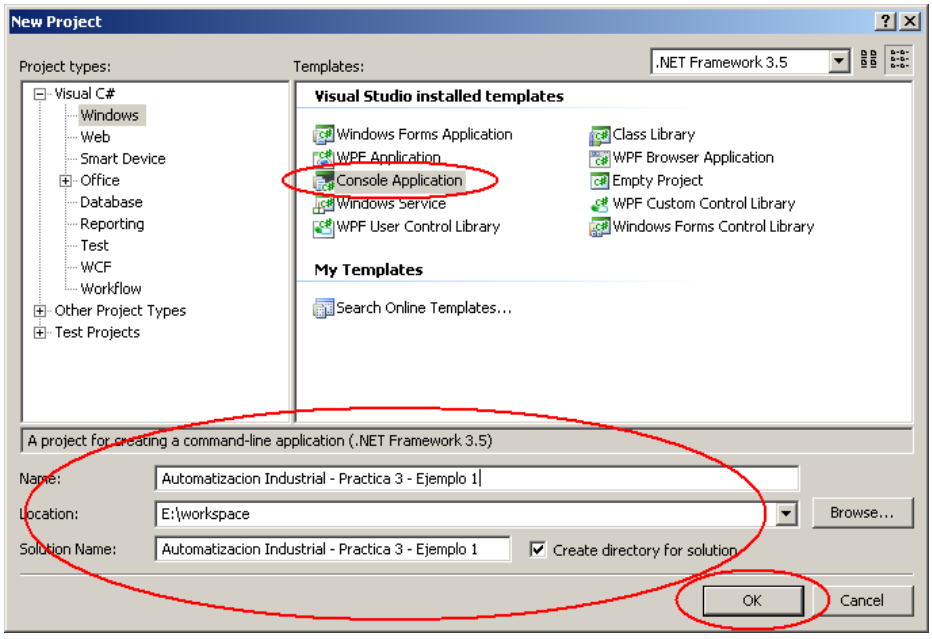

Ilustración 2: ***Ventana de creación de un proyecto***

5. Ya tenemos nuestro proyecto creado (Ilustración 3).

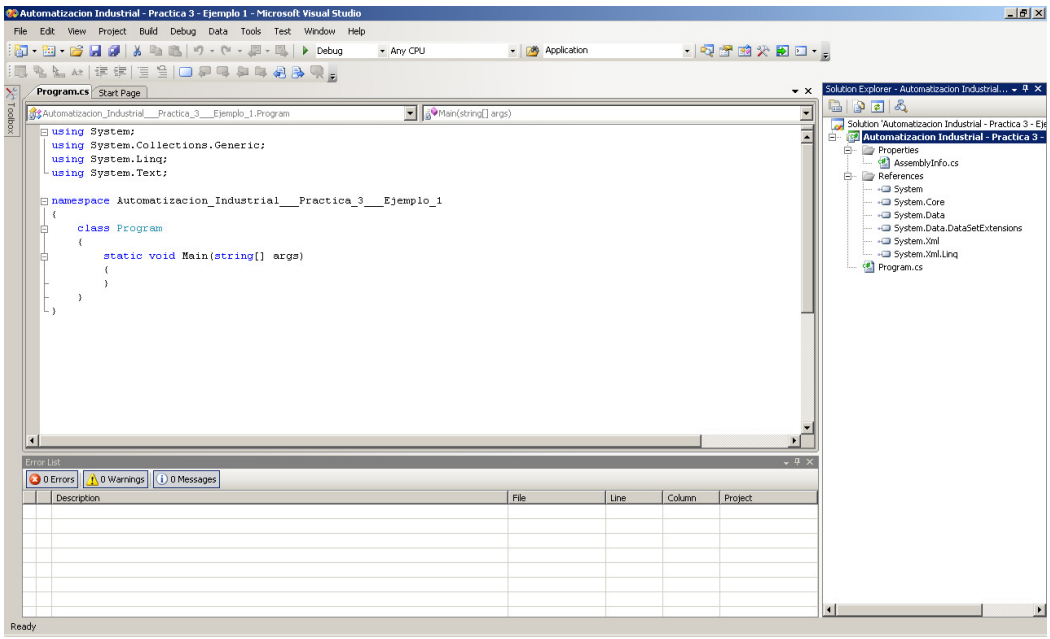

Ilustración 3: *Proyecto nuevo*

6. El siguiente paso es código fuente de la librería FishFa30 en nuestro proyecto. Para ello, movemos el fichero FishFa30.cs al directorio donde hemos guardado el proyecto y la añadimos a la solución que nos ha creado el MVSC#. Esto se hace haciendo clic derecho en la solución de nuestro proyecto y navegando por *Añadir* > *Objeto existente*... (ver Ilustración 4).

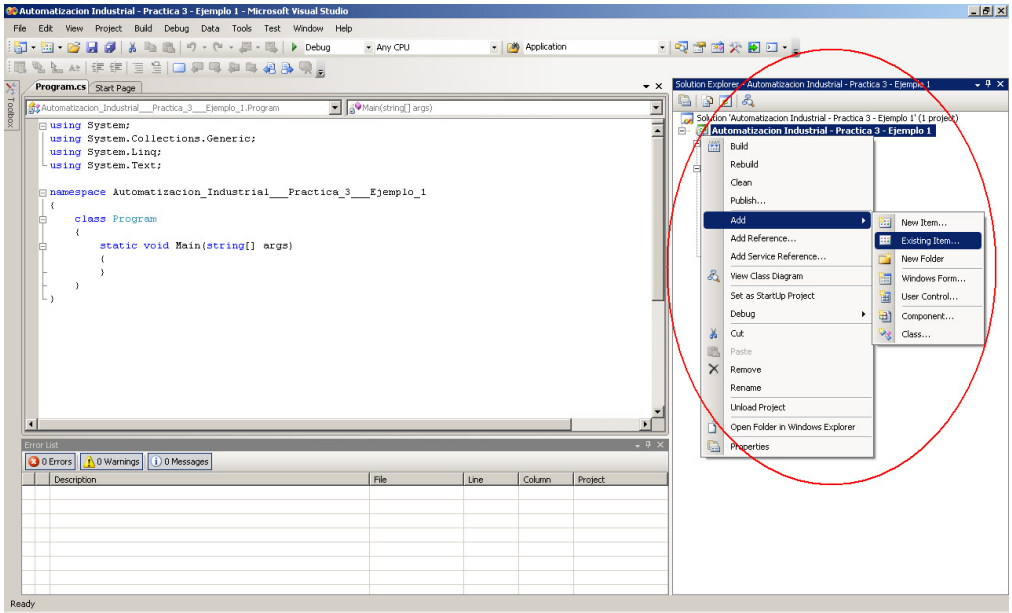

Ilustración 4: ***Añadir una librería existente***

7. En la ventana emergente que nos aparece, debemos buscar el fichero FishFa30.cs que hemos copiado en la carpeta de nuestro proyecto y clicar en *Aceptar*. Como podemos observar, ya sale la librería dentro de nuestro proyecto (ver Ilustración 5).

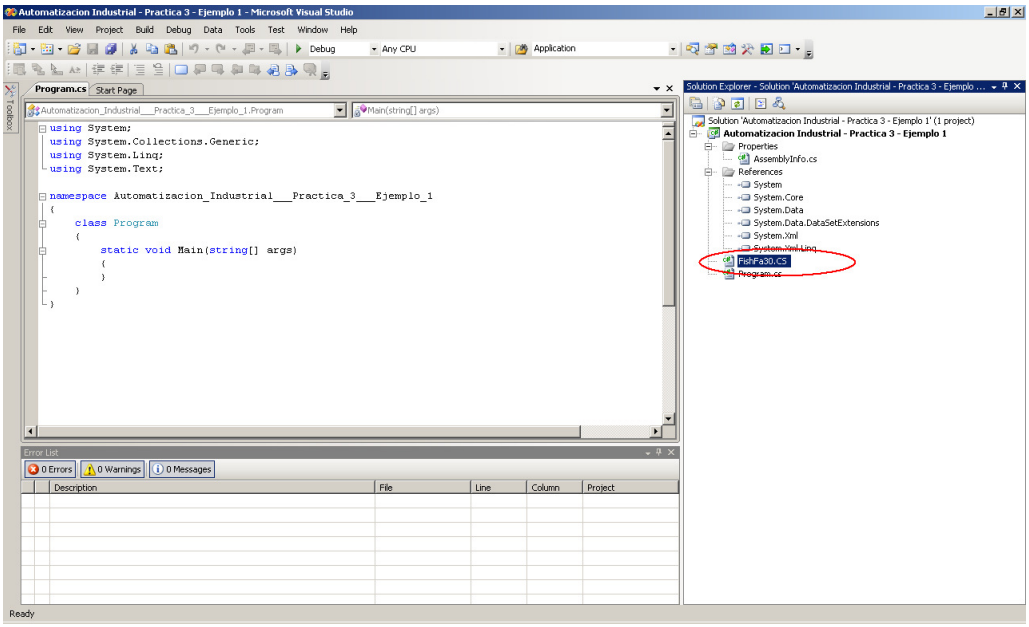

Ilustración 5: *Librería añadida*

8. Por último, es necesario hacer un pequeño ajuste en el entorno para evitar unos errores que aparecen con las ventanas. Consiste en navegar por *Proyecto* > *Añadir referencia*... (ver Ilustración 6).

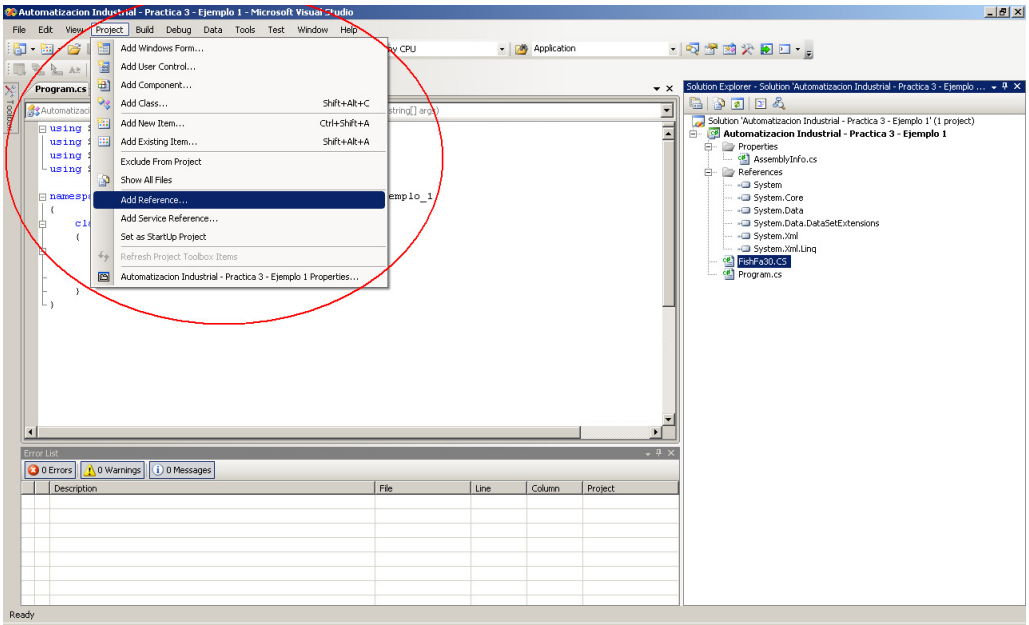

Ilustración 6: ***Añadir una referencia***

9. Y a continuación seleccionamos *System.Windows.Forms* y hacemos clic en *Aceptar* (ver Ilustración 7).

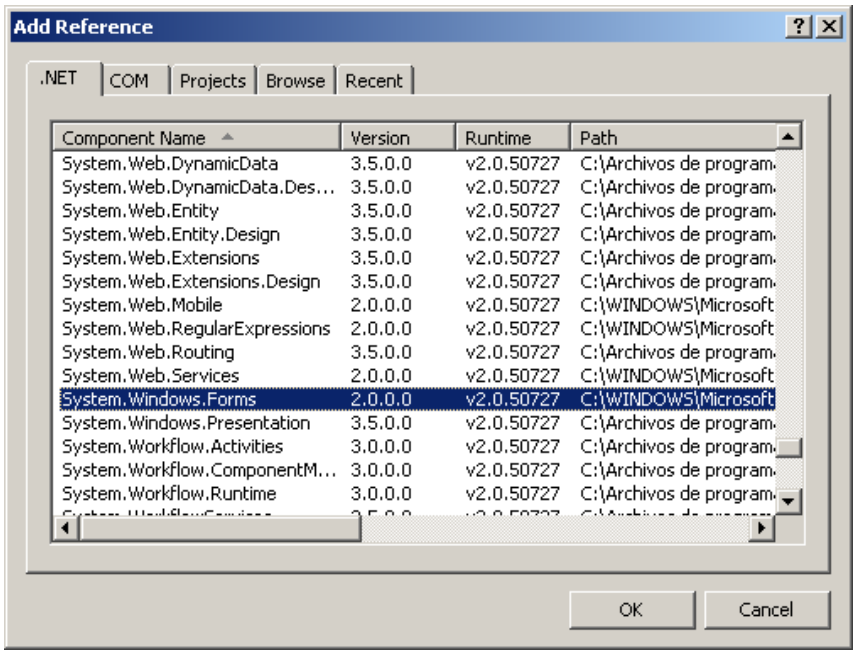

Ilustración 7: *Añadir la referencia System.Windows.Forms*

10. Por último, comprobamos que se ha añadido bien la referencia (ver Ilustración 8) y… ¡a programar!

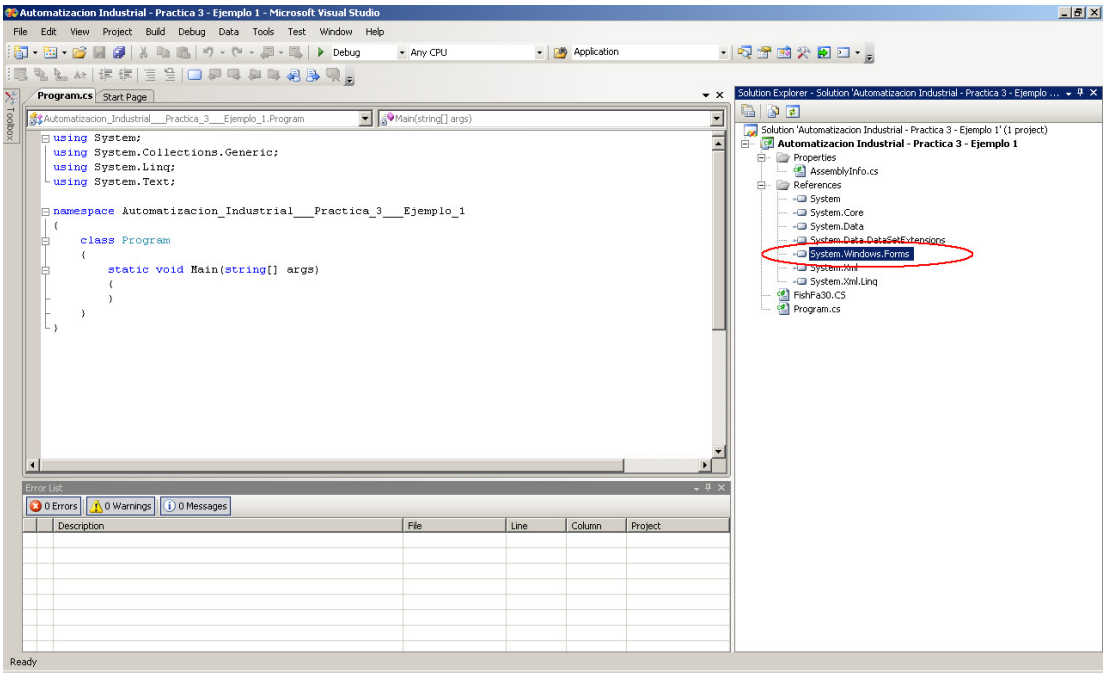

Ilustración 8: ***Referencia System.Windows.Forms añadida***

3.2. Descripción de la librería FishFa30

La clase FishFace será la interfaz por medio de la cual controlaremos la tarjeta inteligente de fishertechnik. Los métodos que utilizaremos de la clase serán las siguientes:

- **void OpenInterface(Port PortNr):** lo utilizaremos para crear la conexión con la tarjeta inteligente. Para ello, tendremos que indicarle el puerto al que tenemos conectada la FTI. Este método no devuelve nada y el control de errores se hace mediante excepciones.

- **void CloseInterface():** cerraremos la conexión con la tarjeta. Este método tampoco devuelve nada, el control de errores mediante excepciones.

- **int GetAnalog(Nr AnalogNr):** este método sirve para consultar el valor de las entradas analógicas. Indicaremos por parámetro la entrada analógica queremos consultar y nos devolverá un entero que representará lo que está leyendo en la entrada en ese momento.

- **bool GetInput(Nr InputNr):** vamos a utilizar este método para consultar el valor de las entradas digitales. Para ello le indicamos qué entrada queremos consultar por parámetro y nos devolverá true si la entrada está a 1 y false si la entrada está a 0.

- **void Pause(int mSek):** este método detendrá la interfaz durante los microsegundos que le indiquemos por parámetro.

- **void WaitForTime(int mSek)**: tiene un comportamiento similar al de la función **Pause(int mSek)**.

- **void SetLamp(Nr LampNr, Dir OnOff):** gracias a este método podremos manejar las lámpara. Éstas se conectan igual que los motores, a una de las salidas digitales de la tarjeta inteligente, pero solamente se podrá indicar si están encendidas o apagadas. Al método le pasaremos por parámetro la salida a modificar y el estado al que queremos

llevarla. Los estados serán unos predefinidos que leeremos del objeto de la clase Enumeration “Dir”.

- **void SetMotor(Nr MotorNr, Dir Direction):** gracias a este método podremos manejar los motores. Éstos funcionan igual que las lámparas, explicadas anteriormente, pero nos permiten un número mayor de posibilidades en cuanto a la elección de la dirección. Podremos hacer que gire en un sentido o en otro, además esto nos permite realizar la multiplexación para las lámparas.

3.3. Ejemplos prácticos

Para poder crear una instancia de la clase FishFa30, y utilizar sus atributos y funciones, debemos declarar de algún modo dónde se encuentran. Para hacer esto hay dos formas:

1. “*Importando”* la librería mediante la sentencia *using*.

```
using FishFa30;
…
FishFace fish = new FishFace();
fish.SetMotor(Nr.M1, Dir.On);
```

Tabla 1: *Referencia a la clase FishFa30 mediante la sentencia "using"*

2. Haciendo referencia a la librería cada vez que vamos a utilizar un atributo o método.

```
//using FishFa30;
…
FishFa30.FishFace fish = new FishFa30.FishFace();
fish.SetMotor(FishFa30.Nr.M1, FishFa30.Dir.On);
```

Tabla 2: *Referencia a la clase FishFa30 sin la sentencia "using"*

3.3.1. Primer ejemplo práctico

En este primer ejemplo se muestra cómo hacer funcionar dos motores, conectados a las salidas digitales M1 y M2, durante 2 segundos.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
//using FishFa30;

namespace Ejemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Creamos una instancia de la clase FishFace
            FishFa30.FishFace fish = new FishFa30.FishFace();
            try
            {
                // Abrimos la conexion con interfaz
                fish.OpenInterface(FishFa30.Port.COM1);
                // Activamos el actuador conectado a la salida M2.
                fish.SetMotor(FishFa30.Nr.M2, FishFa30.Dir.On);
                // Activamos el actuador conectado a la salida M1 a la Derecha
                fish.SetMotor(FishFa30.Nr.M1, FishFa30.Dir.Right);
                // Realizamos un wait para esperar un tiempo, en este caso 2000
                fish.WaitForTime(2000);
                // Desactivamos los actuadores conectados a las salidas M1 y M2
                fish.SetMotor(FishFa30.Nr.M1, FishFa30.Dir.Off);
                fish.SetMotor(FishFa30.Nr.M2, FishFa30.Dir.Off);
                // Por ultimo cerramos la conexion con la interfaz
                fish.CloseInterface();
            }
            catch (FishFa30.FishFaceException)
            {
                Console.WriteLine("ERROR:: No se ha podido abrir la interfaz");
                Console.ReadLine();
            }
        }
    }
}
```

Tabla 3: *Ejemplo práctico 1*

3.3.2. Segundo ejemplo práctico

En este segundo ejemplo se muestra cómo hacer para que un interruptor conectado a la entrada digital E1 ponga en funcionamiento un motor conectado a la salida digital M1.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Ejemplo2
{
    class Program
    {
        static void Main(string[] args)
        {
            Boolean hayEntrada = false;
            // Creamos una instancia de la clase FishFace
            FishFa30.FishFace fish = new FishFa30.FishFace();
            try
            {
                // Abrimos la conexion con el interfaz
                fish.OpenInterface(FishFa30.Port.COM1);
                // Esperamos una señal por parte del sensor conectado a la
entrada digital E1
                while (!hayEntrada)
                {
                    hayEntrada = fish.GetInput(FishFa30.Nr.E1);
                }
                // Una vez que se ha activado el interruptor...
                if (hayEntrada)
                {
                    // Activamos el actuador conectado a la salida digital M1
                    fish.SetMotor(FishFa30.Nr.M1, FishFa30.Dir.On);
                    // Esperamos 3000 ms
                    fish.Pause(3000);
                    // Paramos el actuador conectado a la salida digital M1
                    fish.SetMotor(FishFa30.Nr.M1, FishFa30.Dir.Off);
                }
                // Cerramos la conexion con el interfaz
                fish.CloseInterface();
                Console.ReadLine();

            }
            catch (FishFa30.FishFaceException)
            {
                Console.WriteLine("ERROR:: No se ha podido abrir la interfaz");
                Console.ReadLine();
            }
        }
    }
}
```

Tabla 4: *Ejemplo práctico 2*

4. Conclusiones

Como se ha podido observar, la programación del interfaz FTI es sencilla si se tienen ciertos conocimientos en programación y el lenguaje escogido no supone una barrera aunque no se conozca a fondo, dada su similitud con otros lenguajes muy extendidos como C y Java. En nuestro caso, esta es la primera vez que programamos en C# y apenas hemos tenido problemas. El único problema que nos supuso el entorno es en la referencia a *System.Windows.Forms*, que fue resuelto tal y como se ha descrito en el paso 8 de la sección 3.1 (Cómo crear un proyecto en Microsoft Visual Studio C#).

Además, algo que no es muy agradable y supone un gran impedimento es que toda la documentación referente a la librería *umFish30.dll* y la clase *FishFa30.cs* se encuentra en alemán y hasta ahora no hemos conseguido hacernos de una versión en inglés.

Por último, decir que la realización de los ejemplos proporcionados con esta práctica nos ha aumentado considerablemente el interés por la asignatura dado que es la primera vez que hacemos este tipo de programación, además nos parece muy interesante y divertido.

5. Bibliografía

1: Página de Ulrich-Müller

http://www.ulrich-mueller.de/csecke.htm

2: Wikipedia: La enciclopedia libre

http://es.wikipedia.org/wiki/C_Sharp

http://es.wikipedia.org/wiki/Microsoft_Visual_Studio

3: Página de #develop (Sharp Develop):
http://www.icsharpcode.net/OpenSource/SD/

4: Página oficial de la plataforma Mono:
http://mono-project.com/Main_Page

5: Joseph Albahari; Ben Albahari, *"C# 3.0 in a Nutshell, 3rd Edition"*.