EasyPLC v.5 SDK

Table of Contents

1. **Drivers Development**

2. **Plugins Development**

EasyPLC S.D.K.

This guide helps you to extend the possibilities of the EasyPLC Software Suite.

EasyPLC provides extension tools in order to increase the software capabilities. These tools are known as:

- Drivers.
- Plugins.

The drivers allow managing analogic and/or digital inputs and outputs, and are used to communicate with:

- Any I/O hardware/device compatible with Microsoft Windows.
- Third party software.

The Plugins allow to add new functionalities to your logic programs, such as:

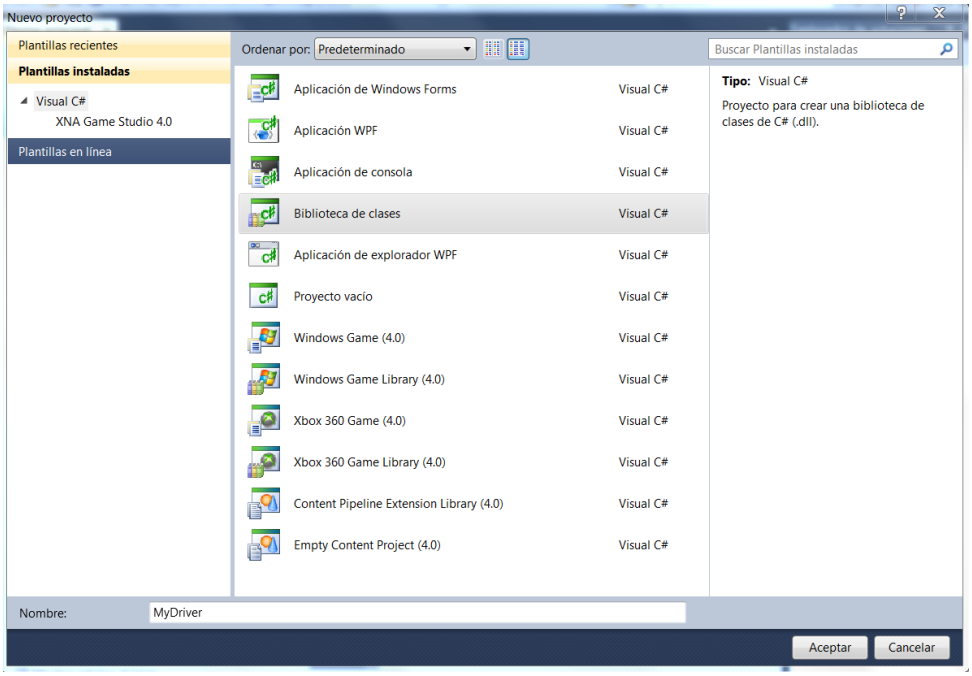
- Communicate with any hardware connected with the PC (graphic displays, data loggers, special keyboards, cameras, etc...).
- Create your own user defined functions.
- And much more...

To create your own Drivers and Plugins you need the free programming tool Microsoft Visual Studio 2010 Express Edition, you can download freely form the Microsoft web site.

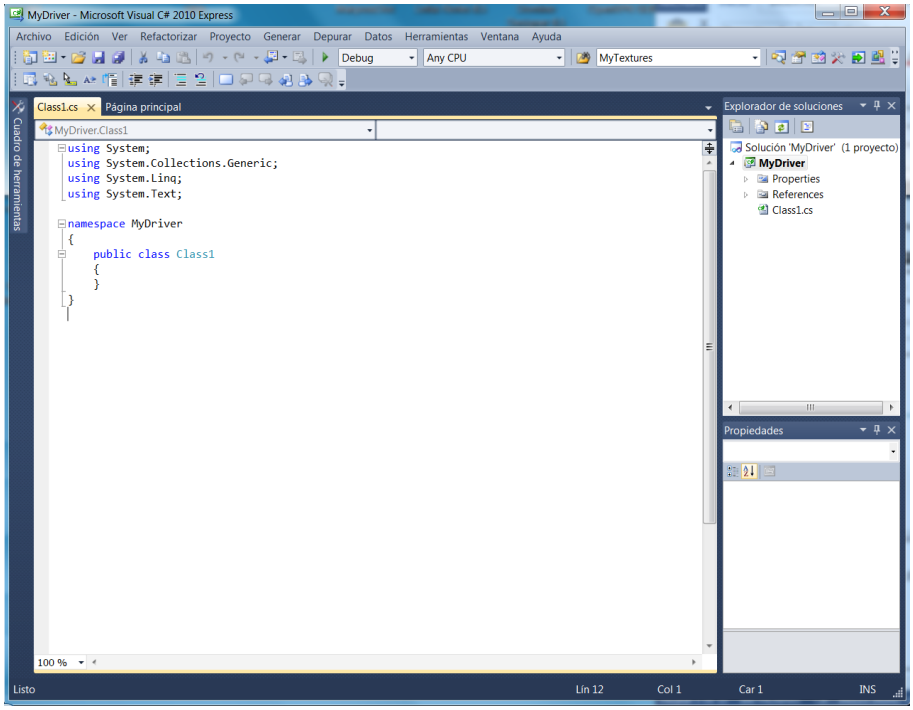
1. Drivers Development

Open Microsoft Visual Studio 2010 Express Edition.

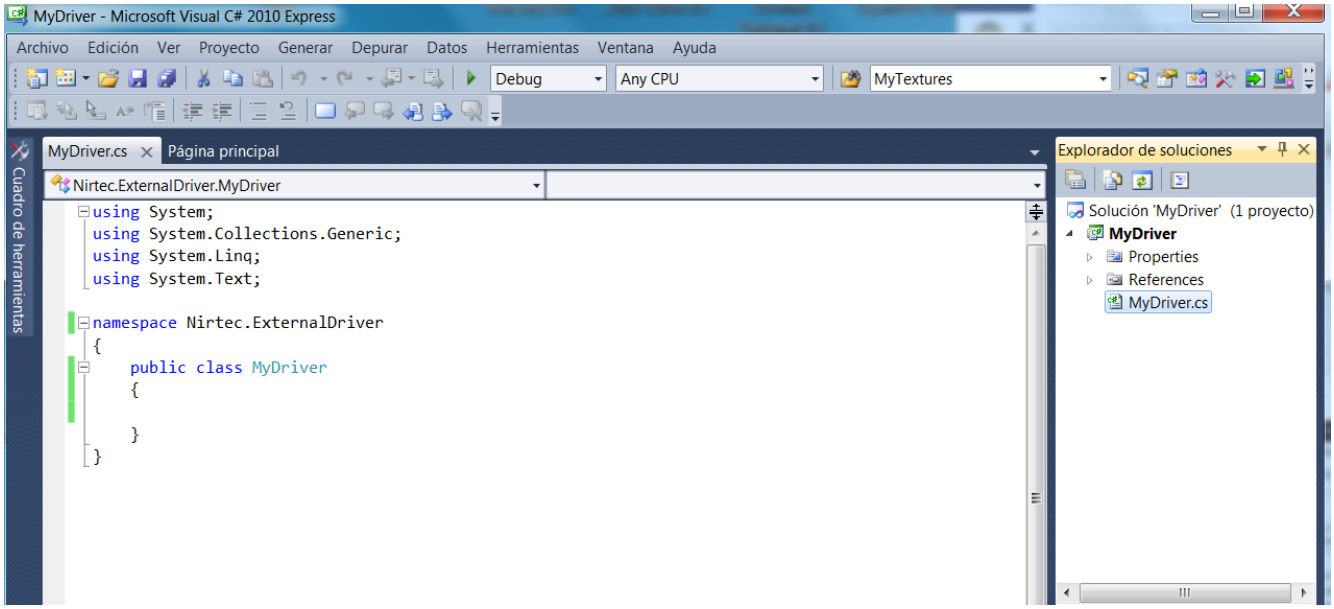
Select File -> New Project. From the New Project window, select Class Library and type your new Driver Name: MyDriver in the example. Click OK.



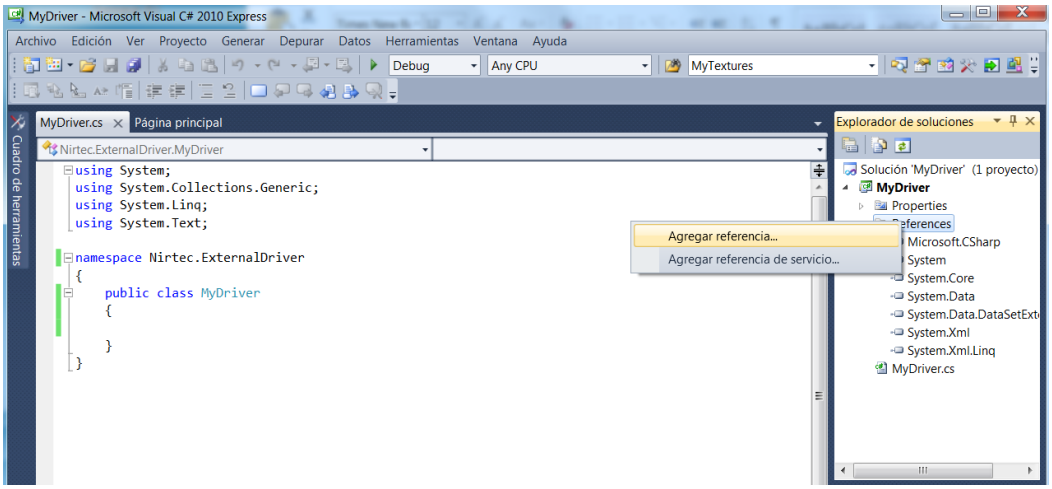
The following template is created automatically:



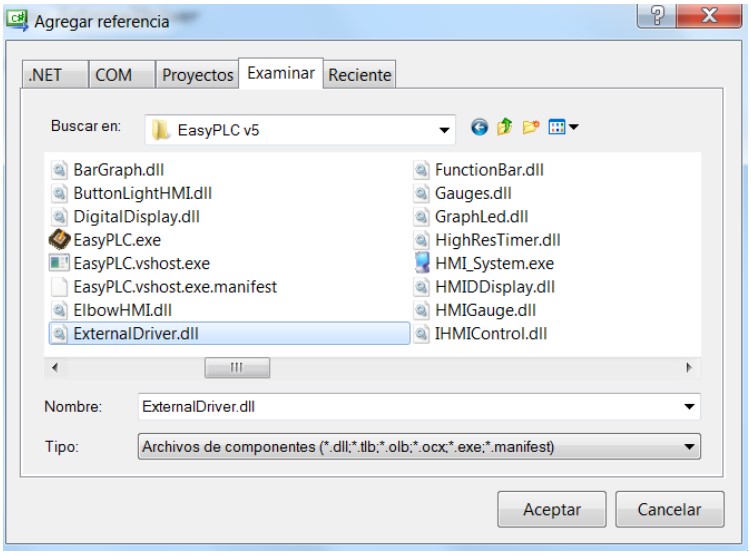
Then change the namespace for Nirtec.ExternalDriver and change the class name and class1.cs file for other more descriptive.



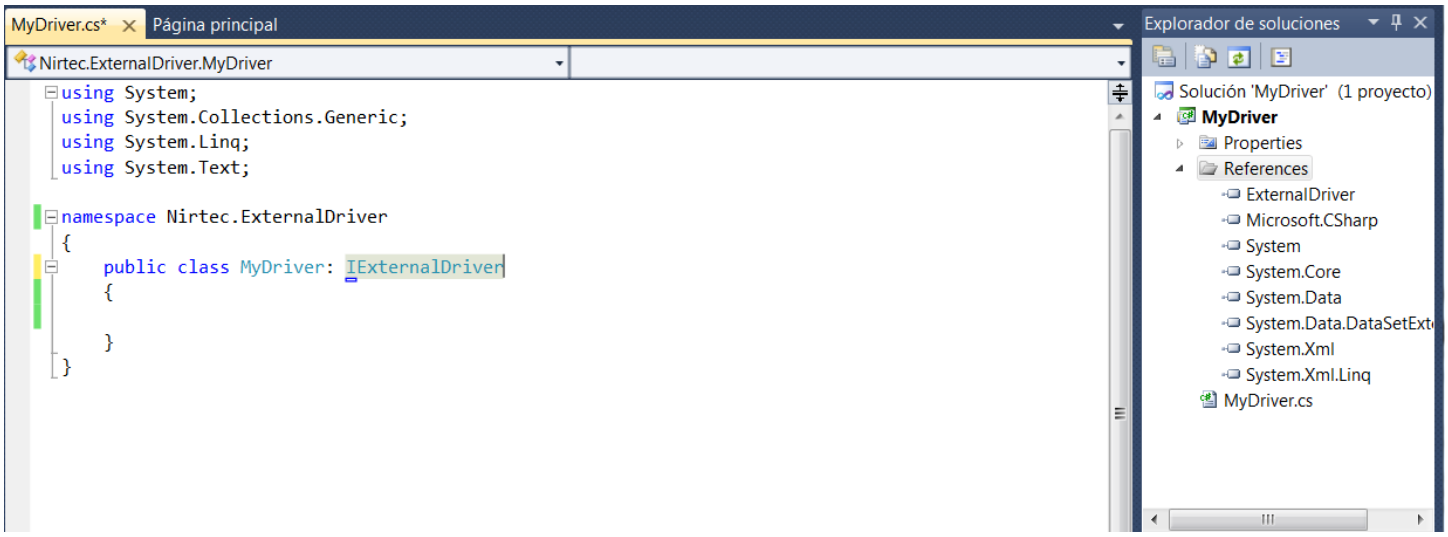
Right click on References node of the Project Explorer tree, and select Add Reference from the contextual menu.



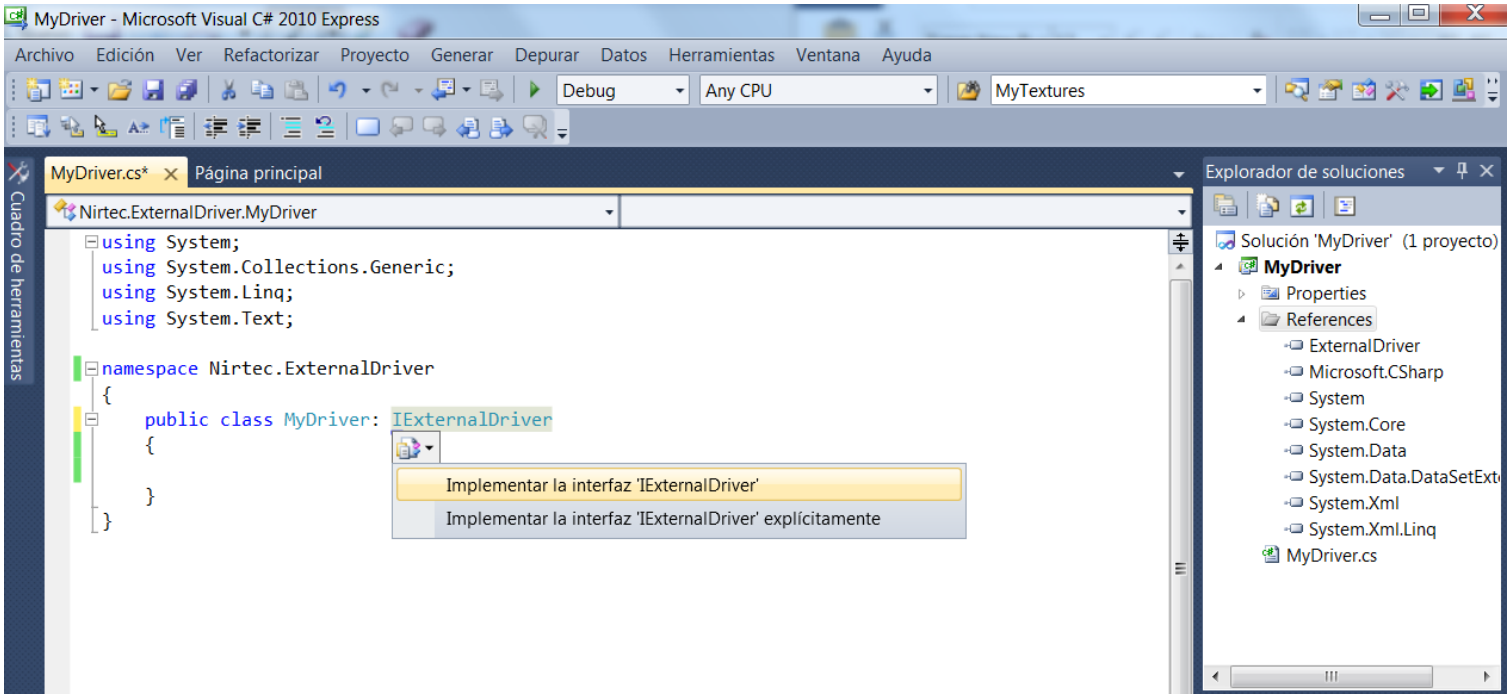
Click on the Examine tab and go to the installed EasyPLC folder, then select the file ExternalDriver.dll



Once the library is added to the project, implement the Interface IExtenalDriver to the class:



Now click on the IExtenalDriver word and select implement the interface 'IExternalDriver'



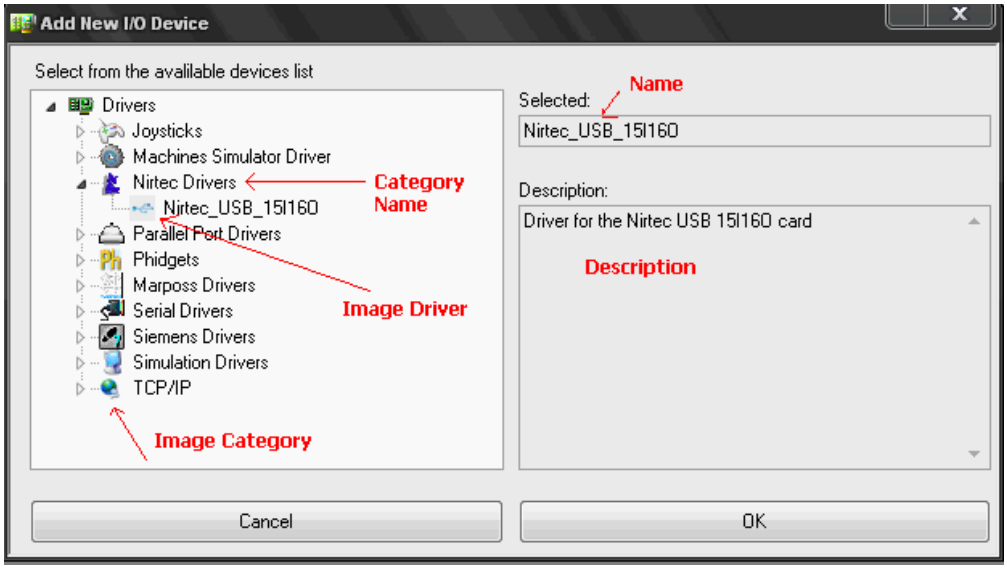
Once each method is programmed correctly, the drivers will correctly implement all methods of operation within the interface.Automatically it will be added all the methods that implements this interface. The following methods may be used with these drivers to correctly program your devices:

```
// Returns true if the driver has analogic I/O otherwise returns false
  public bool AnalogicDevice
  {
      get { throw new NotImplementedException(); }
  }

// This method contains a float array with the analogic input values
  public float[] AnalogicInputs()
  {
      throw new NotImplementedException();
  }

// This method contains a float array with the analogic output values
  public float[] AnalogicOutputs()
  {
      throw new NotImplementedException();
  }

// This method is written by EasyPLC to send to the driver the values of analogic
// outputs
  public void AnalogicOutputs(float[] value)
  {
      throw new NotImplementedException();
  }

// Returns the category name of the driver (see Window Driver picture)
  public string Category
  {
      get { throw new NotImplementedException(); }
  }

// Event launched when user finish the driver configuration
  public event EventHandler ConfigChanged;

// Method called when user press the Driver configure button
  public void Configure()
  {
      throw new NotImplementedException();
  }

// Returns the category name of the driver (see Window Driver picture)
  public string Description
  {
      get { throw new NotImplementedException(); }
  }

// Returns true if the driver has digital I/O otherwise returns false
  public bool DigitalDevice
  {
      get { throw new NotImplementedException(); }
  }

// This method contains a boolean array with the digital input values
  public bool[] DigitalInputs()
  {
      throw new NotImplementedException();
  }
```

```
//This method contains a boolean array with the digital output values
  public bool[] DigitalOutputs()
  {
      throw new NotImplementedException();
  }

//This method is written by EasyPLC to send to the driver the values of digital
// outputs
  public void DigitalOutputs(bool[] value)
  {
      throw new NotImplementedException();
  }

// Method called when the PLC pass to stop mode and closes the communication with
// the driver.
  public void Finish()
  {
      throw new NotImplementedException();
  }

// Not used
  public string IdValue
  {
      get { throw new NotImplementedException(); }
  }

// Returns the category name of the driver (see Window Driver picture)
  public System.Drawing.Icon ImageCategory
  {
      get { throw new NotImplementedException(); }
  }
// Returns the category name of the driver (see Window Driver picture)
  public System.Drawing.Icon ImageDriver
  {
      get { throw new NotImplementedException(); }
  }

// Method called by EasyPLC when starts the communication with the driver
  public void Init()
  {
      throw new NotImplementedException();
  }

// Returns a string showing the current driver status.
  public string MessageStatus
  {
      get { throw new NotImplementedException(); }
  }
// Returns the category name of the driver (see Window Driver picture)
  public string Name
  {
      get { throw new NotImplementedException(); }
  }

// Method called by EasyPLC when opens the communication with the driver, is passed
// like parameters the driver number (defined in the Hardware - > I/O devices
// EasyPLC Editor node) and an ArrayList with the parameters configured in the
// Configure Method.
  public void OpenDriver(int driverNumber, System.Collections.ArrayList args)
  {
```

```
            throw new NotImplementedException();
        }
    // EasyPLC will write in this method the current PLC status, true for PLC in run
    // mode, false for PLC in Stop mode
        public void PLCStatus(bool status)
        {
            throw new NotImplementedException();
        }
    // EasyPLC will call this method in the beginning of the scan cycle in order to
    // read the analogic inputs of this driver
        public void ReadAllAnalogicInputs()
        {
            throw new NotImplementedException();
        }

    // EasyPLC will call this method in the beginning of the scan cycle in order to
    // read the digital inputs of this driver
        public void ReadAllDigitalInputs()
        {
            throw new NotImplementedException();
        }

    // Returns the driver status, true when is ready to work.
        public bool Ready
        {
            get { throw new NotImplementedException(); }
        }

    // EasyPLC will call this method when closes the communication with the driver
        public void ResetAnalogicOutputs()
        {
            throw new NotImplementedException();
        }

    // EasyPLC will call this method when closes the communication with the driver
        public void ResetDigitalOutputs()
        {
            throw new NotImplementedException();
        }

    // Returns the total analogic inputs of this driver
        public int TotalAnalogicInputsNumber
        {
            get { throw new NotImplementedException(); }
        }

    // Returns the total analogic outputs of this driver
        public int TotalAnalogicsOutputNumber
        {
            get { throw new NotImplementedException(); }
        }

    // Returns the total digital inputs of this driver
        public int TotalDigitalInputsNumber
        {
            get { throw new NotImplementedException(); }
        }

    // Returns the total digital outputs of this driver
        public int TotalDigitalOutputsNumber
        {
```

```
        get { throw new NotImplementedException(); }
    }

    // EasyPLC will call this method at the end of the scan cycle in order to write the
    // analogic outputs of this driver
    public void WriteAllAnalogicOutputs()
    {
        throw new NotImplementedException();
    }

    // EasyPLC will call this method at the end of the scan cycle in order to write the
    // digital outputs of this driver
    public void WriteAllDigitalOutputs()
    {
        throw new NotImplementedException();
    }
```



EasyPLC window Driver

EasyPLC would make the following calls when directing the PLC to RUN mode:

ExternalDriver.OpenDr(int driverNumber, string[] args)

ExternalDriver.Init

Do

Loop While Not ExternalDriver.Ready

In configuration mode, when the user presses the button ‘Configure’ EasyPLC makes the following calls:

ExternalDriver.OpenDr(int driverNumber, string[] args)

ExternalDriver.Configure ()

The PLC scan mode cycle works in the following way:

1. For each Driver read the status of all theirs analogic & digital inputs:
    - `ReadAllAnalogicInputs()`
    - `ReadAllDigitalInputs()`
2. Execute the logic program.
3. For each Driver write the status of all theirs analogic & digital outputs:
    - `WriteAllAnalogicOutputs()`
    - `WriteAllDigitalOutputs()`

This behavior is repeated cyclically each PLC scan mode cycle during the PLC is in Run mode.

Once the driver library is compiled, it must be copied in the EasyPLC Drivers folder.

IMPORTANT NOTE: the user created drivers only works with the EasyPLC registered version, (not allowed in DEMO version).

2. Plugins Development

Open Microsoft Visual Studio 2010 Express Edition.

Select File -> New Project. From the New Project window, select Class Library and type your new plugin name. Click OK.

Change the namespace and class names for a one that describes their operation (the namespace and class names must be the same).

The plugin library must implement the following method called Help:

```
public Dictionary<string,string> Help()
{
      Dictionary<string, string> dict = new Dictionary<string, string>();

      return dict;
}
```

And returns a Dictionary object that contains as elements as methods this plugin have. Each dictionary pair is composed by a key (the method name) and by a value (the information about this method).

Also a Finish method must to be implemented, that will be called by EasyPLC when closes the communication with it (when the PLC pass to stop mode).

Once the plugin library is compiled, the created library must to be placed in the EasyPLC Plugins folder.

Now is showed a plugin example:

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

namespace Test
{
    public class Test
    {

        public Dictionary<string,string> Help()
        {
            Dictionary<string, string> dict = new Dictionary<string, string>();
            dict.Add("Sum", "Return the addition of the two passed numbers");
            dict.Add("Pot2", "Raises the passed number to the power potence");
            dict.Add("Div", "Divides the two passed numbers");
            dict.Add("Half", "Divides the passed number by 2");
            dict.Add("Message", "Shows a message");
            return dict;
        }
        public void Finish()
        {
        }

        public int Sum(int num1, int num2)
        {
            return num1 + num2;
        }

        public long Pot2(int num)
        {
            return num * num;
        }

        public float Div(float num1, float num2)
        {
            return num1 / num2;
        }

        public double Half(double num)
        {
            return num / 2;
        }

        public void Message(string txt)
        {
            MessageBox.Show(txt);
        }

    }
}
```

IMPORTANT NOTE: the Plugins only works with the EasyPLC registered version, (not allowed in DEMO version).

Idea, Design & Programming by Rafael Izquierdo
Valencia (Spain)

www.nirtec.com