

Maria Servo Controller Library Tutorial

Project's website: www.gmiotto.net/servocontrol.aspx

Introduction:

Maria Servo Controller is a piece of hardware to control servo motors via USB port. To communicate with the hardware (and control the servos) you can: 1 -Download a ready-to-use program at project's website, or; 2 - Build your own program using a C#.NET library. The library is freely available at the same website.

This tutorial guides you through the steps necessary to make your first program for Maria Servo Controller using the library.

Requirements:

1 - To follow this tutorial you must have already build your controller board. If you don't have build it yet you can find everything you need at the project's webpage. In this page is freely available for download: schematics, board layout, the .hex file (for the microcontroller) and instructions to make your card.

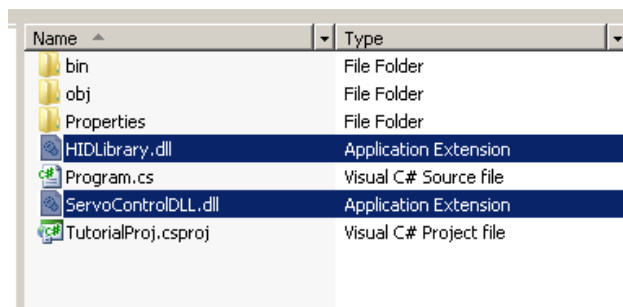
2 – It is recommended for this tutorial that you have two servos connected to your board. Nevertheless, this is not mandatory. You can get the idea with just one servo.

3 – You must have Visual Studio installed and also a basic knowledge of the language C#.NET.

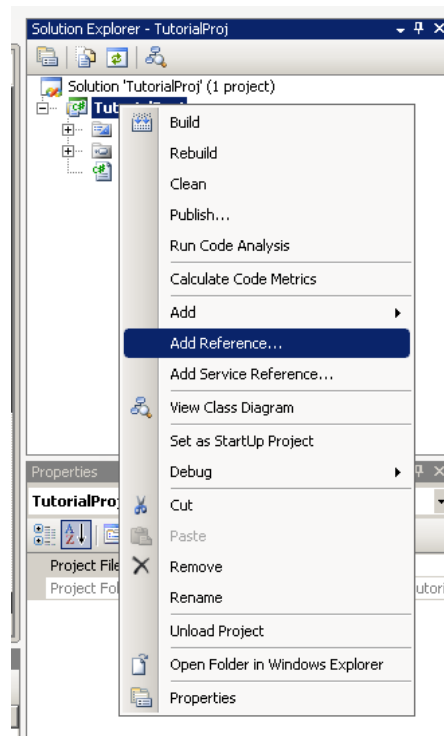
4 – You must have downloaded the library. It is available at the website.

Steps:

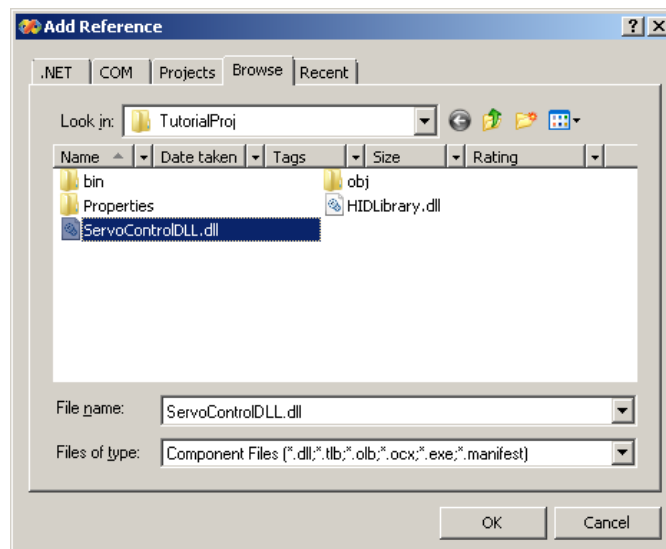
1 – Open Visual Studio and create a new project. Unzip the library file at any location you want. I recommend you to unzip it at the project's folder. The zip file contains two dll files: ServoControl.dll and HIDLibrary.dll. They are both necessary.



2 – Switch to Visual Studio and add references to the library in your project. To do so, right-click at the project name in the project explorer and then click at "Add Reference...".



3 - The following window will open. Click at *Browse*, and select the ServoControl.dll and click *OK*. There is no need to reference HIDLibrary.



4 – Try the following code.

```
using System;
using ServoControlDLL;

namespace TutorialProj
{
    class Program
    {
        static void Main(string[] args)
        {
            ServoController Controller = new ServoController(2);
            Controller.Connect();

            Controller.Servos[0].MoveToAngle(30);
            Controller.Servos[1].MoveToAngle(150);
            Controller.Update();

            System.Threading.Thread.Sleep(5000);

            Controller.Servos[0].MoveToAngle(150);
            Controller.Servos[1].MoveToAngle(30);
            Controller.Update();
        }
    }
}
```

Executing this program will have the following result:

- 1st – Servos 0 and 1 will start to move at the same time. Servo 0 will go to position 30 degrees and servo 1 will go to position 150 degrees.
- 2nd – There will be a pause of five seconds.
- 3rd – Then servos exchange positions. Servo 0 goes to position 150 degrees and servo 1 goes to 30 degrees.

Code explained:

Line 2:

```
using ServoControlDLL;
```

I added this line just to shorten function calls.

Line 10:

```
ServoController Controller = new ServoController(2);
```

This line creates an instance of the class ServoController and configures it to work with two servos.

Note: When using this constructor, the ServoController instance (in our case Controller) shall be able to handle any servo that has the same PWM configuration as a standard Futaba servo. But, notice that Maria can be configured to use any kind of PWM servo. But this would require some additional lines of code.

Line 11:

```
Controller.Connect();
```

The function Connect() set up the communication link between PC and the controller.

Lines 13 and 14:

```
Controller.Servos[0].MoveToAngle(30);  
Controller.Servos[1].MoveToAngle(150);
```

These two lines tell what will be the position of the servos in the next move.

Line 15:

```
Controller.Update();
```

Up to this point, servos didn't move. Just when function Update() is called commands are sent to and servos move. So, the role of function Update() is to send to hardware all moves requested by MoveToAngle() functions.

Line 17:

```
System.Threading.Thread.Sleep(5000);
```

This line, as you probably know, does not belong to Maria's library. It just makes your program "stop" for 5000 milliseconds.

Basic routine:

