Serializer™ .NET Library Installation & QuickStart Guide

Summerour Robotics Corporation, www.roboticsconnection.com, 2004-2010.

Serializer™ .NET Library Installation & QuickStart Guide v3.0

RoboticsConnection provides a free .NET library to allow customers to easily and quickly interface their high level applications (using the .NET framework) with the Serializer Robot Controller.

This guide provides some simple steps to install and configure the .NET library, and add it into a sample Visual Studio 2005, 2008, or 2010 Project (including Express Editions).

The installer will install the library in the proper directories, and make sure that it's available through the Global Assembly Cache (GAC).

Note: *The Serializer™ Library is based on the Microsoft .NET 2.0 Framework.*

Installing the Library:

1.) Download the latest version of the Serializer™ .NET Library Installer from the Docs\Downloads tab on the Serializer Robot Controller page:

http://www.roboticsconnection.com/p-16-serializer-robot-controller-new-version-30.aspx

2.) Run the installer, and perform the following steps:

Figure 1 – Serializer™ Library Installation Start

Figure 2 – Serializer™ Library Installation Folder

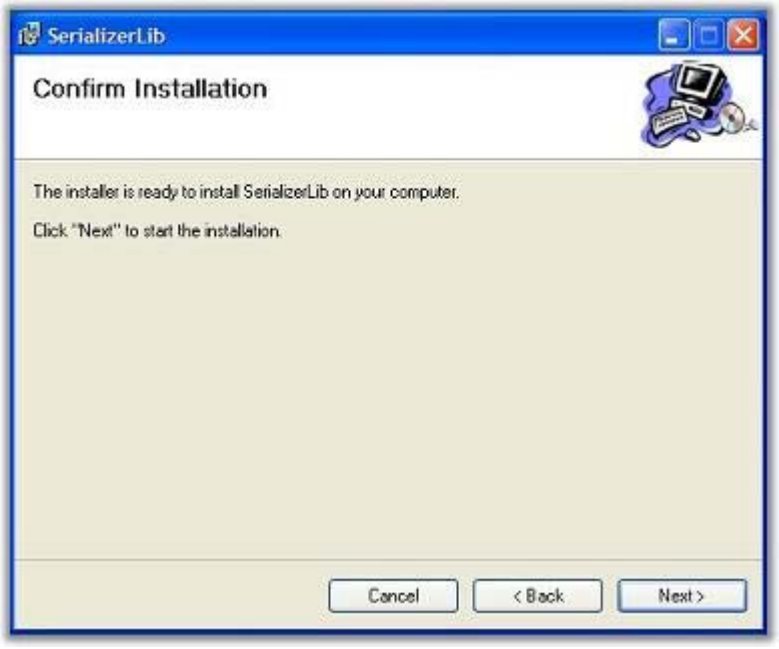

Figure 3 – Serializer™ Library Installation Confirmation

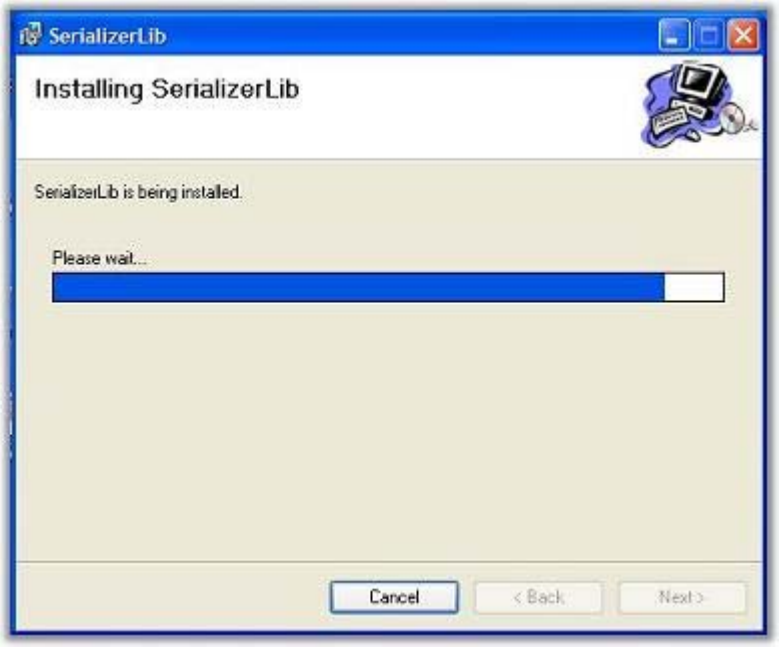

Figure 4 – Serializer™ Library Installation Progress Bar

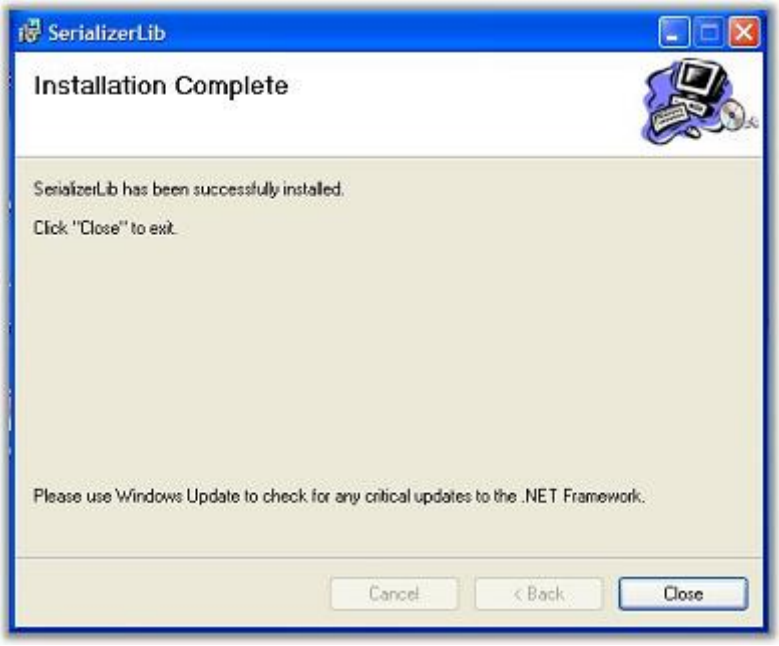

Figure 5 – Serializer™ Library Installation Complete

Using the Serializer .NET Library:

Now that the Serializer™ library assemblies have been installed, they are available for use.

The Serializer™ library is built for the full .NET Framework, as well as the Compact .NET Framework. Likewise, there is an assembly (dll file) for each version. ***It is important that you select the proper reference assembly for your type of application.***

Once you create an application in Visual Studio, you can add a reference to the Serializer™ .NET library assemblies in the "*Solution Explorer Menu*" by right clicking on the references folder, and selecting "*Add Reference*".

If your application uses the regular .NET Framework then select:

SerializerLib

If your application uses the Compact Framework (e.g. a Windows CE application) then select the assembly pertaining to the Compact Framework.

SerializerLib.CompactFramework

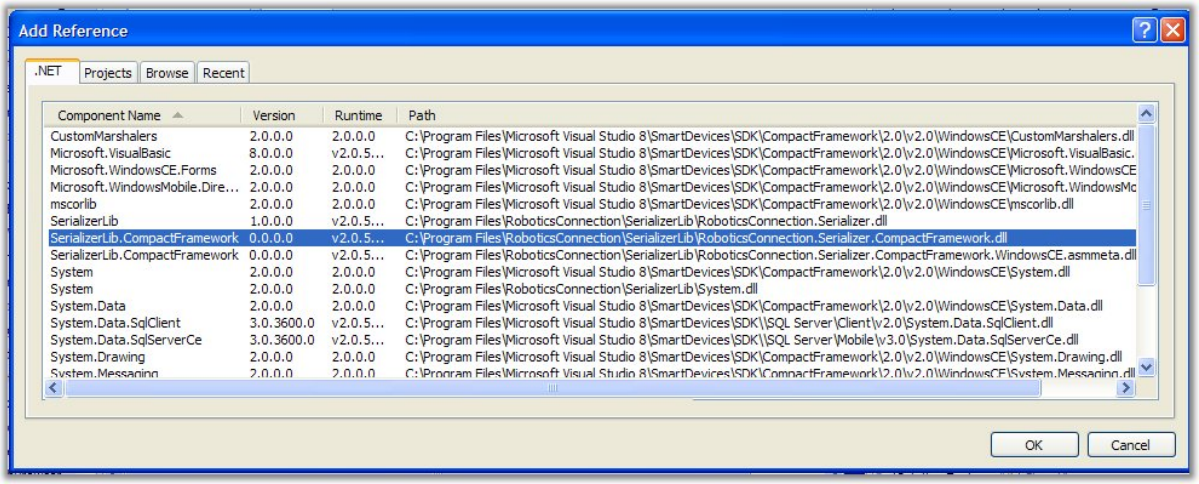

Figure 6 – Visual Studio Add Reference Dialog

Creating a project:

The following example adds the Serializer™ library as a reference within a Visual Studio project, and uses it in a VERY basic application.

Create a new project in Visual Studio. For this example, we're creating a C# Device Application based on the .NET Compact Framework for a Windows CE 5.0 "Smart Device", which will be called "*SerializerTestApp*". If you were creating an application using the full .NET framework, you would follow the exact same steps, except that you would use the other assembly file (SerializerLib).

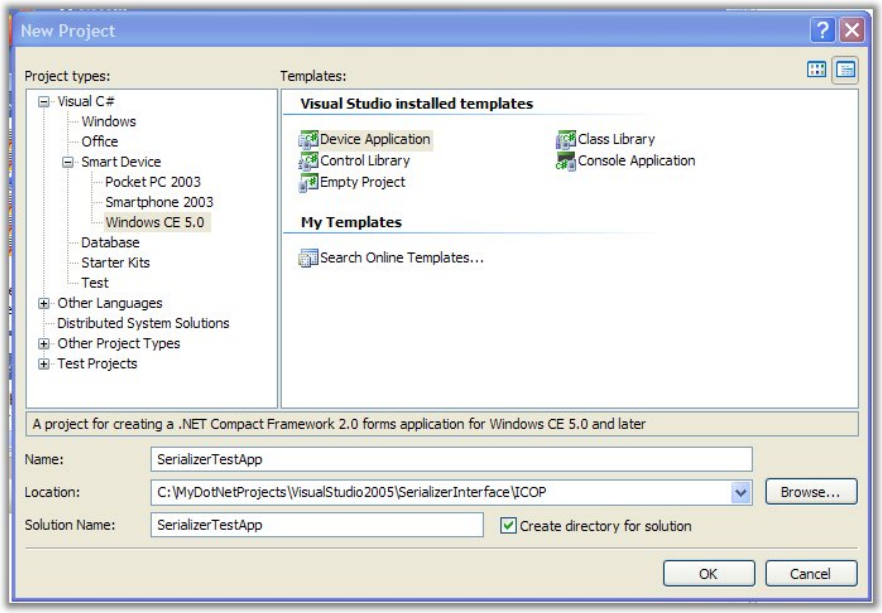

Figure 7 – Creating a Windows CE 5.0 Smart Device Application

Add a reference to the appropriate Serializer™ assembly, by right clicking on the highlighted "References" folder (as shown below), and selecting "Add Reference".

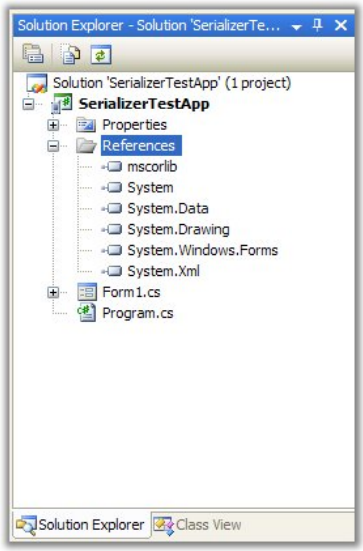

Figure 8 – Solution Explorer – References folder

The following dialog box will appear displaying the available assemblies that are available to the application. Scroll down until you see the available assemblies for the SerializerLib.

Since we're developing a Win CE application, select the "SerializerLib.CompactFramework" which corresponds to the RoboticsConnection.Serializer.CompactFramework.dll (highlighted).

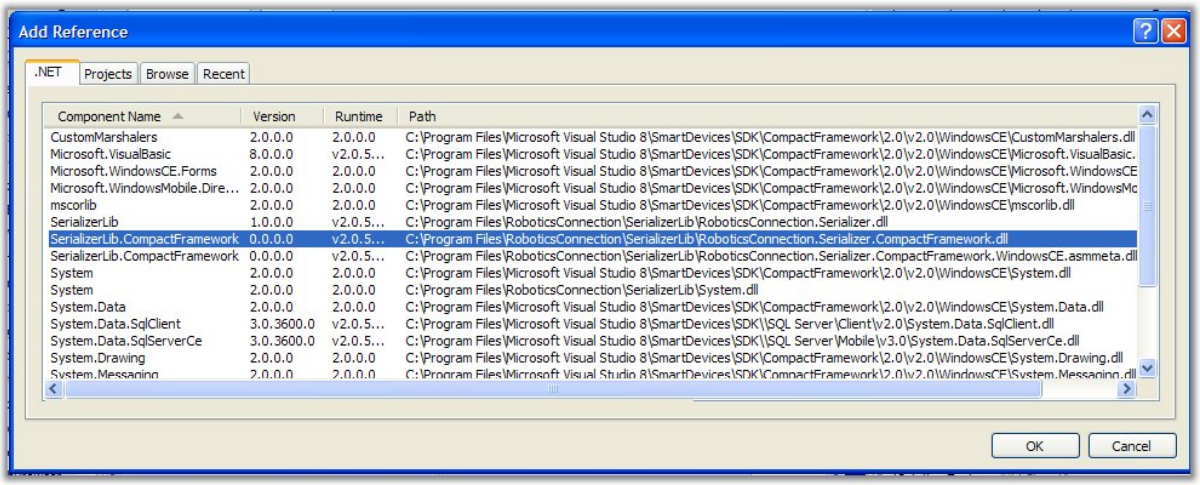

Figure 9 – SerializerLib.CompactFramework Add Reference Selection

Next, add the correct "using" namespace entries in the C# files. You'll want to include all of the using namespace entries for RoboticsConnection below in both the Form1.cs class, as well as in the Form1.cs[Design].

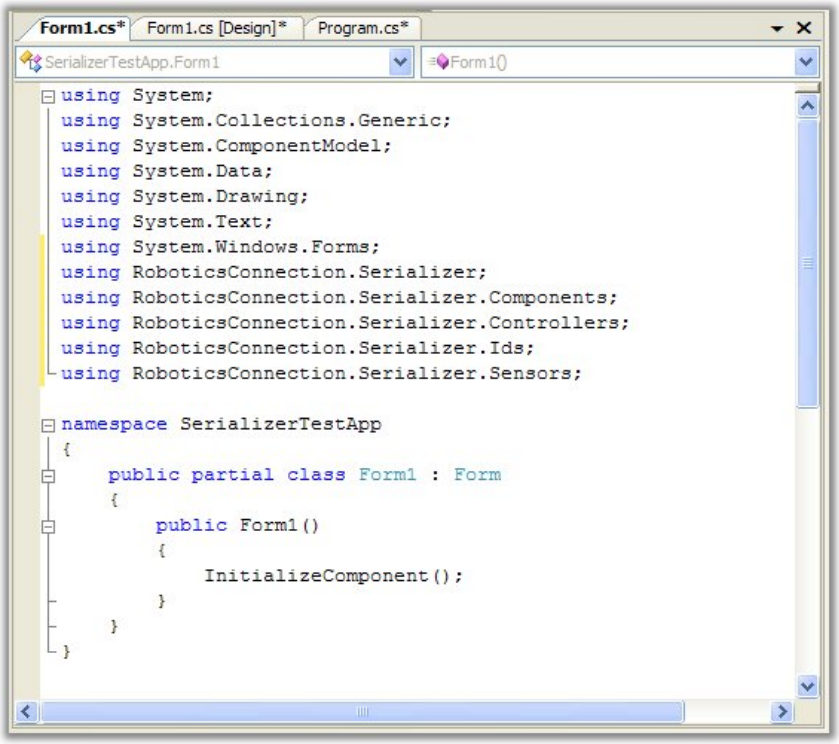

Figure 10 – Adding RoboticsConnection Namespaces

At this point, we've added a reference to the SerializerLib assembly for the .NET Compact Framework, and added the appropriate using namespaces so that all of the Serializer™ components can be used.

We're now ready to add the appropriate initialization code for the Serializer™. In the file "Form1.Designer.cs" file, add the following code:

```
#region SERIALIZER OBJECTS:
// Serializer Objects:
private static Serializer s_;
#endregion
```

Figure 11 – Serializer Object

Also, add the following method (in the same file), which will get invoked to initialzie the Serializer™.

```
#region SERIALIZER INITIALIZATION
private void InitSerializerComponents()
{
    // Initialize Serializer:
    s_ = new Serializer();
    s_.PortName = "COM2";
    s_.BaudRate = 19200;
}
#endregion
```

Figure 12 – InitSerializerComponents() Method

In the Form1.cs file, add the following code in the Form1() constructor:

```
using RoboticsConnection.Serializer;
using RoboticsConnection.Serializer.Components;
using RoboticsConnection.Serializer.Controllers;
using RoboticsConnection.Serializer.Ids;
using RoboticsConnection.Serializer.Sensors;

namespace SerializerTestApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            InitSerializerComponents();

            // Inform library to start communicating with
            // the Serializer board.
            s_.StartCommunication();
        }
```

Figure 13 – Serializer initialization invocation.

The call to InitSerializerComponents() will create the Serializer™ object, and initialize a few Properties, such as baud rate, and COM port.

The call to s_.StartCommunication() instructs the SerializerLib to start communicating with the Serializer™ board using the Serializer's Serial Protocol (defined in the Serializer User Guide).

Thus, when the application starts, and Form1 is constructed, all of the Serializer™ Objects, and communication with the Serializer™ board will be initialized.

Now we need to Drag a Timer object (highlighted below) from the Toolbox over to the man Form. This Timer object will be used to periodically handle events generated by the SerializerLib.

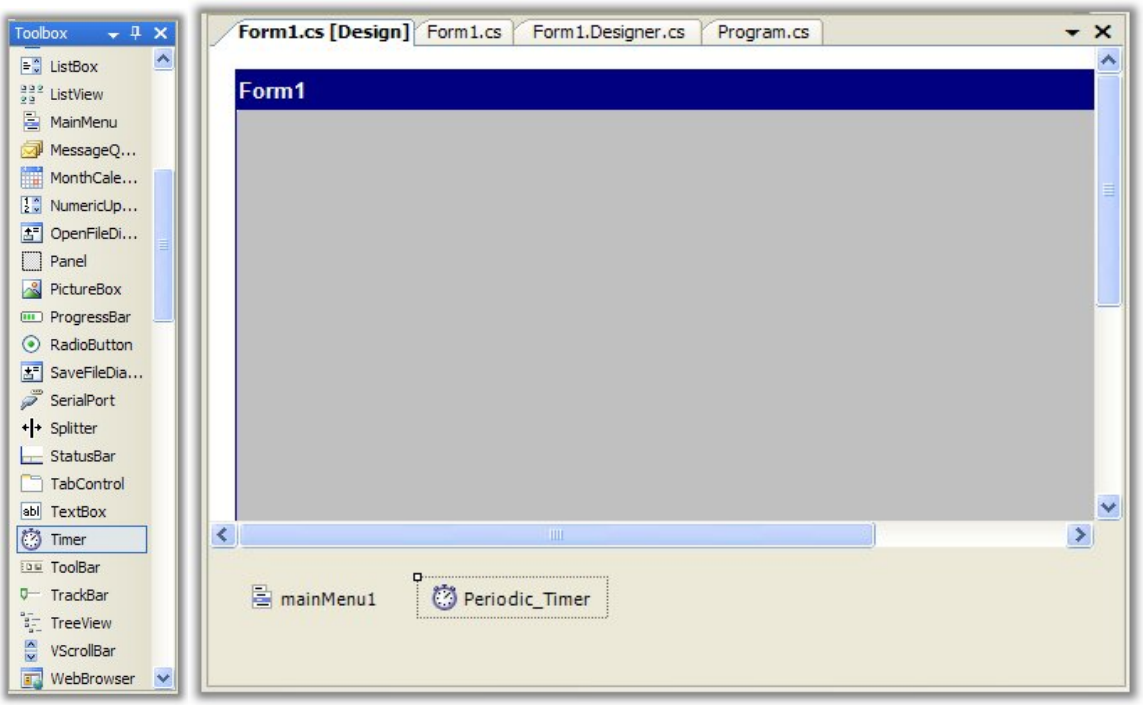

Figure 14 – Dragging Timer Toolbox Component to Form1

The Timer object should appear below the main form pane. Clicking on the Timer object should bring up the Properties dialog box in Visual Studio.

Change the Name of the object to “Periodic_Timer” as shown below:

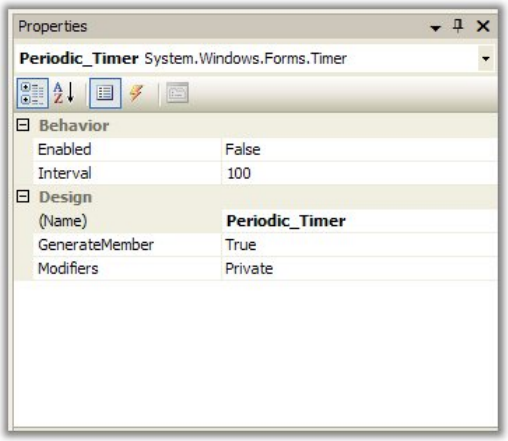

Figure 15 – Periodic_Timer Properties

You should now see the “Periodic_Timer” name reflected in the object name that you just clicked under the Form pane. NOTE: The Interval defaults to 100, which means a Periodic_Timer event will occur every 100msec. MAKE SURE you set the “Enabled” property to “True”, else the timer callback handler you’re about to add won’t get called. *You could also set the “Enabled” proper to “True” from within a Serializer CommunicationStarted Event handler (see example apps at the link below to see how to do this). That way, you don’t start “Pumping Events” until you know that the application has properly connected to the Serializer.*

Double click the “Periodic_Timer” object in the main form pane. This will bring up the “Form1.cs” file, and the “Periodic_Timer_Tick()” event handler will automatically be inserted in the Form1 class. Within the Periodic_Timer_Tick() event handler method, add the code shown below:

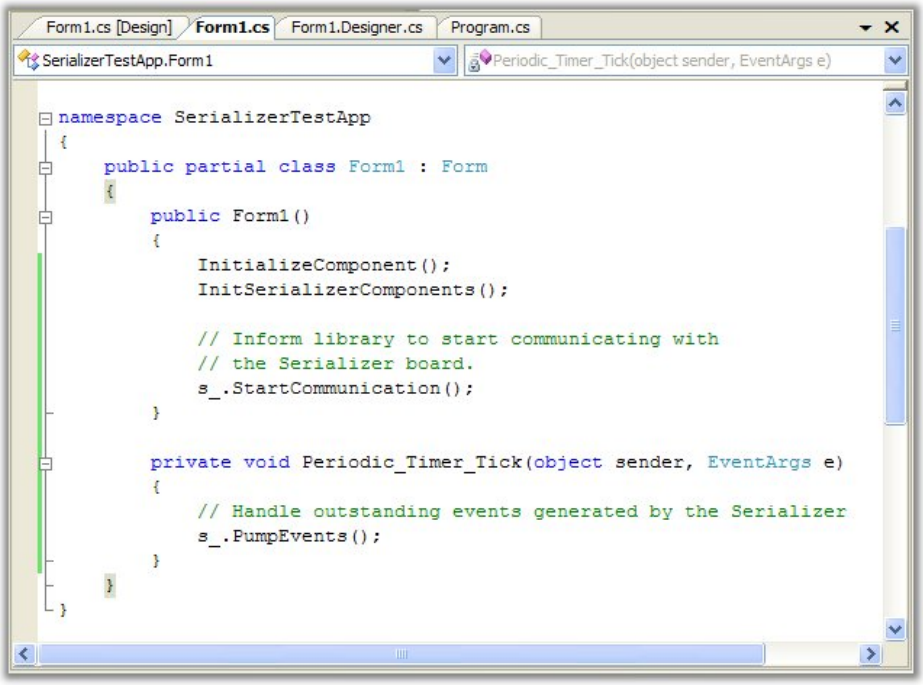

Figure 16 – Adding s_.PumpEvents() to the Periodic Timer Event Handler

The purpose of the PumpEvents() invocation within the Periodic Timer callback allows the internals of the SerializerLib to perform work. PumpEvents() services the events generated internally by the SerializerLib. These events occur based on responses received over the serial link from the Serializer™ board. Each event has an associated callback assigned to it (to perform work), and that callback is invoked by PumpEvents().

Since, in this specific example, the application is spinning in the System.Windows.Form run loop, waiting for events to occur (from buttons being pressed, etc). Therefore, we need the timer to periodically invoke PumpEvents().

The Serializer™ initialization code is now set up. Serializer™ objects can be added as needed.

To continue the example, we're going to add code to blink the two green onboard LEDs. This will consist of adding two checkboxes, one for each LED, a textbox to allow users to enter the blink rate, and a button to allow the user to send the blink command to the Serializer™.

Add two checkbox components by dragging them from the Toolbox onto Form1. Name them "Led1CheckBox" and "Led2CheckBox" in the Properties window. Also, change the labels to "LED1" and "LED2", respectively.

Drag a textbox component by dragging it from the Toolbox onto Form1. Name the textbox "BlinkRateTextBox" in the Properties window.

Drag a button component by dragging it from the Toolbox onto Form1. Name the button "BlinkLEDButton" in the Properties window. Also, change the label to "Blink LEDs".

Drag a label over to the blink rate textbox as shown, and change it so that it reads "LED Blink Rate (0-254)". Your form should now look like the one in the image below:

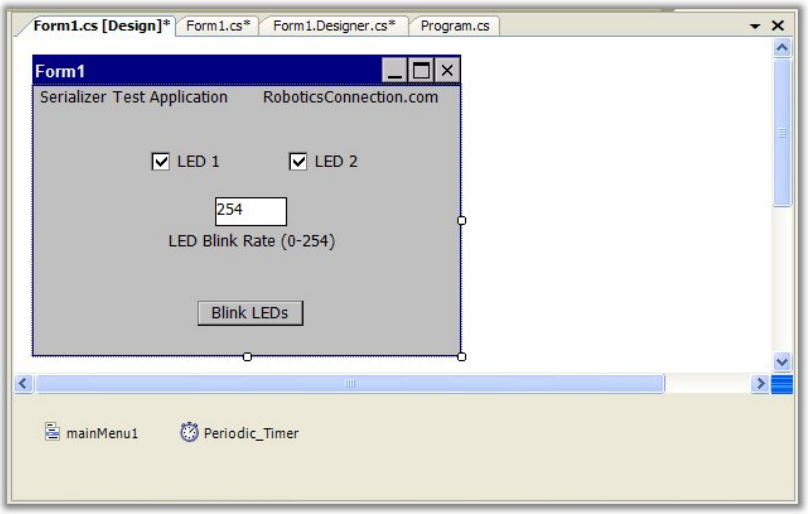

Figure 17 – Final Serializer Text App Form Layout

Double click the BlinkLEDButton, and Form1.cs file should appear, as well as a new "BlinkLEDButton_Click()" event handler as shown below:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        InitSerializerComponents();

        // Inform library to start communicating with
        // the Serializer board.
        s_.StartCommunication();
    }

    private void Periodic_Timer_Tick(object sender, EventArgs e)
    {
        // Handle outstanding events generated by the Serializer
        s_.PumpEvents();
    }

    private void BlinkLEDButton_Click(object sender, EventArgs e)
    {

    }
}
}
```

Figure 18 – BlinkLEDButton_Click() Event Handler

Add the following code to the BlinkLEDButton_Click() event handler method:

```
    private void BlinkLEDButton_Click(object sender, EventArgs e)
    {
        if (Led1CheckBox.Checked)
        {
            s_.BlinkLED(LedId.LED1, Convert.ToInt16(BlinkRateTextBox.Text()));
        }
        if (Led2CheckBox.Checked)
        {
            s_.BlinkLED(LedId.LED2, Convert.ToInt16(BlinkRateTextBox.Text()));
        }
    }
}
```

Figure 19 – Adding Work Code to BlinkLEDButton_Click() Event Handler

The response to the BlinkLEDButton_Click event is to invoke the appropriate SerializerLib.BlinkLED() method on the "checked" LEDs.

At this point, you should be able to build the application, and execute it. Once you make sure you have a Serializer™ connected via a serial cable, and powered up, you can experiment setting various blink rates for each LED. If an LED is unchecked, the blink rate will not be sent to the Serializer™ for that LED id. If you want to stop both LEDs, set a blink rate of 0, and make sure both LEDs are checked before hitting the "Blink LEDs" button.

Serializer™ .NET Library Installation & QuickStart Guide v3.0

Example Projects:
We have many other Serializer sample projects here:

http://www.roboticsconnection.com/t-RoboticApplications.aspx

and we invite you to download and peruse the apps.

Serializer .NET Library Documentation:
Additionally, we have VERY EXTENSIVE documentation for the Serilaizer .NET lib here:

http://www.roboticsconnection.com/multimedia/libraries/MSDNDocumentation

Serializer .NET Library Questions/Suggestions:
If you have any questions or suggestions, please post questions on our forums, specifically in the Serializer .NET Library section. You can also easily search the forums for answers to your specific question.

http://www.roboticsconnection.com/userForums/