Tutorial - Hello World

Spirit Du

Ver. 1.1, 25th September, 2007

Ver. 2.0, 7th September, 2008

Ver. 2.1, 15th September, 2014

Contents

About This Document

In this tutorial, two topics are introduced: (1) how to create a C# project in Microsoft Visual Studio, and (2) how to use Buttons, TextBox, and other controls in the Form. In the window programming course, we use C# as our programming language and Windows Form technology to build window applications. Microsoft Visual Studio is a powerful IDE for Windows and we will use it during the entire semester. Note that the usages of the Button, TextBox, and TableLauputPanel described in this tutorial will be used in the first homework.

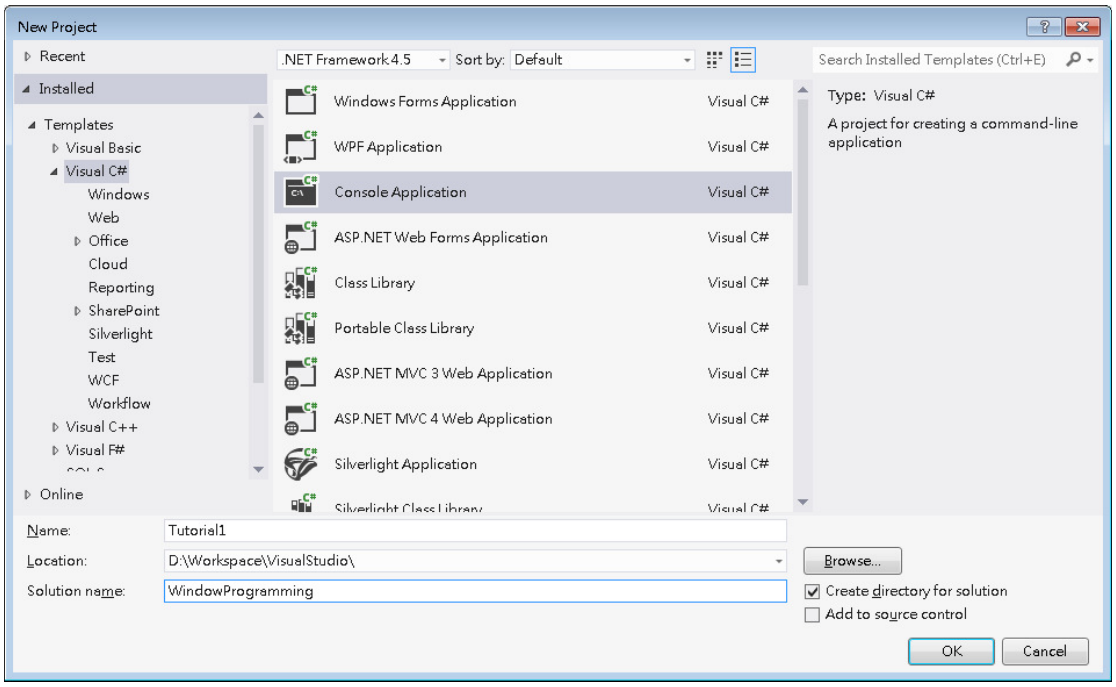
A Hello Message Box

Step 1 Start up Visual Studio

Start → All Programs → Microsoft Visual Studio 2012 → Visual Studio 2012

Step 2 Create a new empty project

File → New → Project, in the New Project dialog (Figure 1). Input the project and solution name.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Forms Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
ASP.NET MVC 3 Web Application Visual C#
ASP.NET MVC 4 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a command-line application

Name: Tutorial1

Location: D:\Workspace\VisualStudio\

Solution name: WindowProgramming

☑ Create directory for solution

☐ Add to source control

Figure 1 New Project Dialog

Step 3 Add reference: System.Windows.Forms

In C#, a reference is similar to a library in other languages. A reference: System.Windows.Forms is needed to create Windows Form applications, Right click on the "References" item (Figure 2) in Solution Explorer, and then choose "Add Reference" in the context menu. Select the reference in the dialog shown in Figure 3, then chick OK.

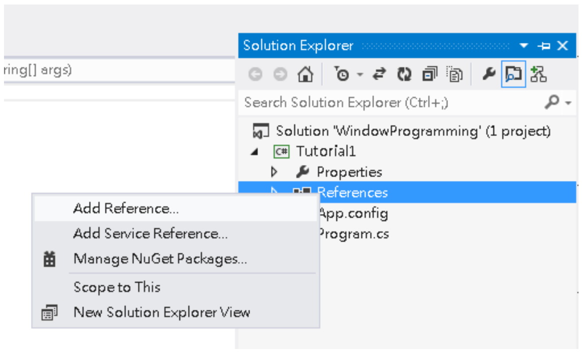
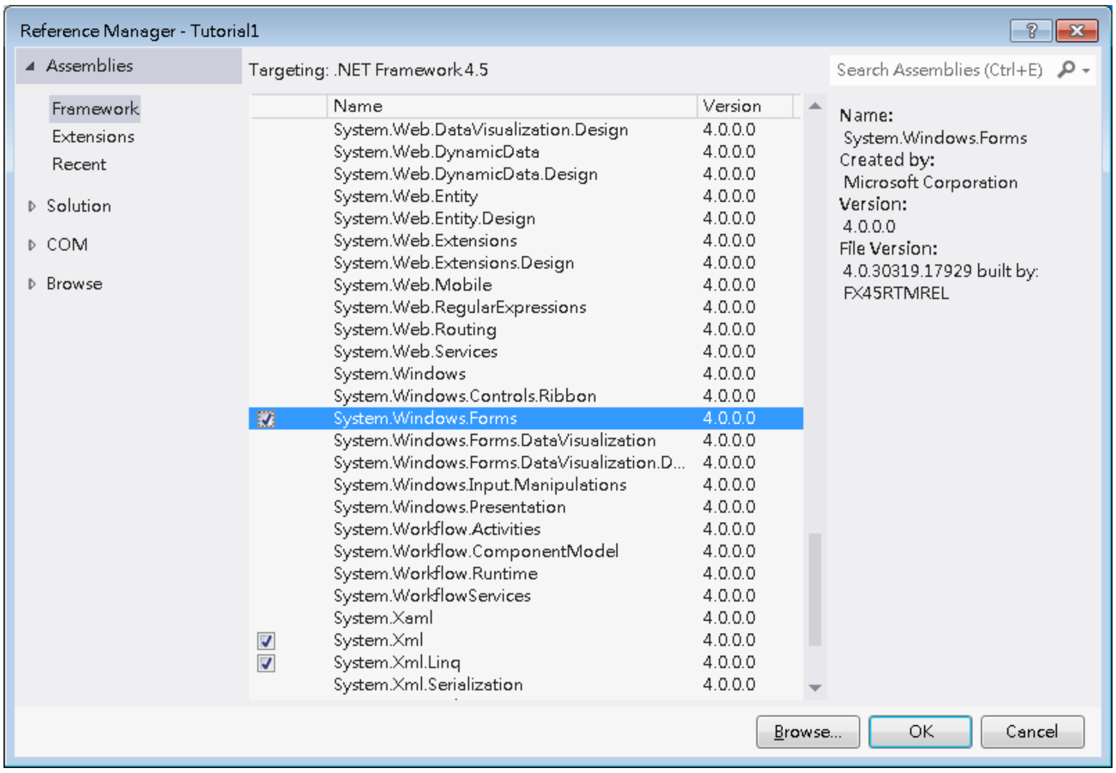


Figure 2 Add references into the project


Reference Manager - Tutorial1

- Assemblies
  - Framework
  - Extensions
  - Recent
- Solution
- COM
- Browse

Targeting: .NET Framework 4.5

Search Assemblies (Ctrl+E)

	Name	Version
	System.Web.DataVisualization.Design	4.0.0.0
	System.Web.DynamicData	4.0.0.0
	System.Web.DynamicData.Design	4.0.0.0
	System.Web.Entity	4.0.0.0
	System.Web.Entity.Design	4.0.0.0
	System.Web.Extensions	4.0.0.0
	System.Web.Extensions.Design	4.0.0.0
	System.Web.Mobile	4.0.0.0
	System.Web.RegularExpressions	4.0.0.0
	System.Web.Routing	4.0.0.0
	System.Web.Services	4.0.0.0
	System.Windows	4.0.0.0
	System.Windows.Controls.Ribbon	4.0.0.0
✓	System.Windows.Forms	4.0.0.0
	System.Windows.Forms.DataVisualization	4.0.0.0
	System.Windows.Forms.DataVisualization.D...	4.0.0.0
	System.Windows.Input.Manipulations	4.0.0.0
	System.Windows.Presentation	4.0.0.0
	System.Workflow.Activities	4.0.0.0
	System.Workflow.ComponentModel	4.0.0.0
	System.Workflow.Runtime	4.0.0.0
	System.WorkflowServices	4.0.0.0
	System.Xaml	4.0.0.0
✓	System.Xml	4.0.0.0
✓	System.Xml.Linq	4.0.0.0
	System.Xml.Serialization	4.0.0.0

Name:
System.Windows.Forms
Created by:
Microsoft Corporation
Version:
4.0.0.0
File Version:
4.0.30319.17929 built by: FX45RTMREL

Figure 3 Add the **System.Windows.Froms** reference

Step 4 Let's write programs

Double click the "Program.cs" item in the Solution Explorer. In the editor, just add a line of code inside the Main() method.

```
System.Windows.Forms.MessageBox.Show("Hello, Windows Forms");
```

Step 5 Say Hello

Press "F5" key. Visual Studio will compile the program and then execute it if there is no error. A message box will be displayed on the screen like Figure 4.

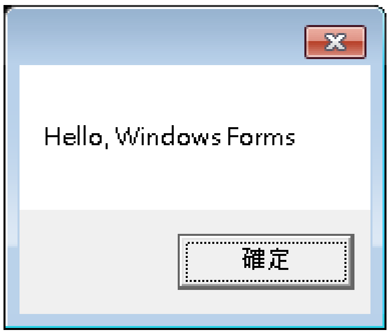
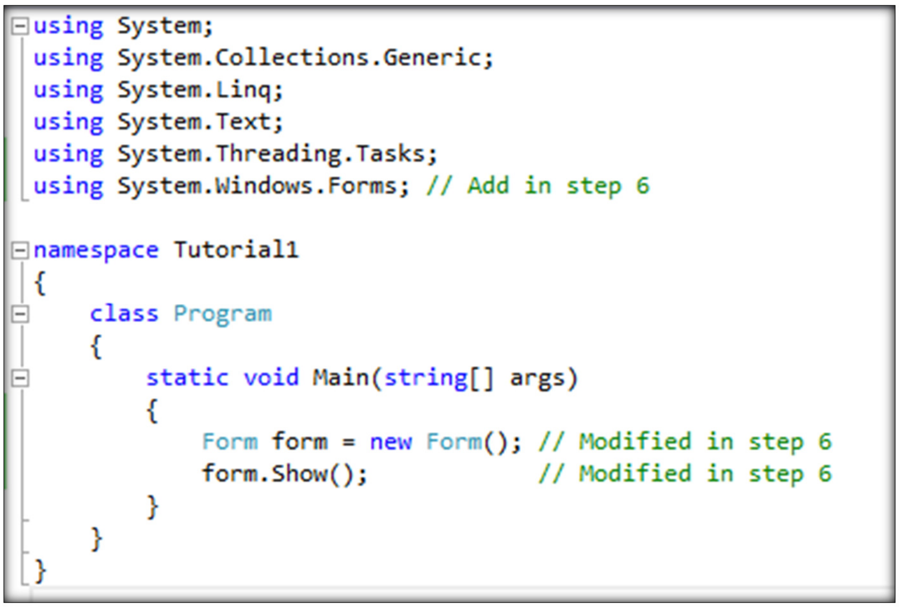

Hello, Windows Forms

Figure 4 A Hello message box

A Hello World Form

Step 6 A flash

Now, it's time to upgrade from a message box to a form, the one actually used in applications. Modify the codes as following, and execute the program to see what happened…

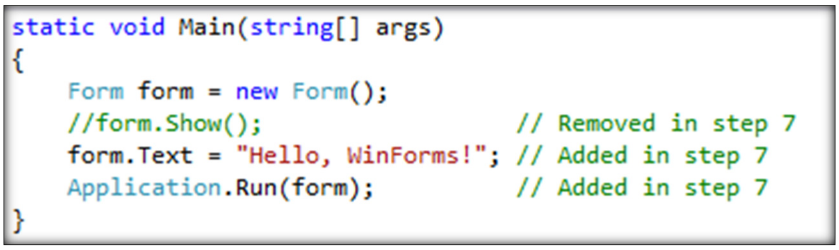


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms; // Add in step 6

namespace Tutorial1
{
    class Program
    {
        static void Main(string[] args)
        {
            Form form = new Form(); // Modified in step 6
            form.Show();            // Modified in step 6
        }
    }
}
```

Step 7 Enduring as the universe

Is it a bug? It seems nothing happened. The program actually calls form.show(), but it doesn't work. Why? Well, it works. It changes the visibility property of the form instance, but the program exits as soon as the statement is executed, so the form disappeared. In order to keep the form until users want to close the form, it needs an additional class: Application to handle the form. Modify the codes as following, and execute the program.



```
static void Main(string[] args)
{
    Form form = new Form();
    //form.Show();                 // Removed in step 7
    form.Text = "Hello, WinForms!"; // Added in step 7
    Application.Run(form);         // Added in step 7
}
```

Step 8 Add a button

After Step 7, a form (Figure 5) is ready, but it can't do anything interesting. Now, modify the codes in Main() as the following to combine the form and the message box. Here, the "button.Click" in line A is a list of event handlers that will be notified when someone click the button. So the line wraps a method "ButtonClicked" as an event handler object, similar to function point in C/C++, and then adds it into the list using the operator: +=. The "Controls" in line B is a container which keeps everything displayed in the form. This line adds the button into the form. Therefore, execute the program and click the button. A message box will be displayed on the screen, when ButtonClicked method is

invoked by runtime because the method is added into the event handler list.

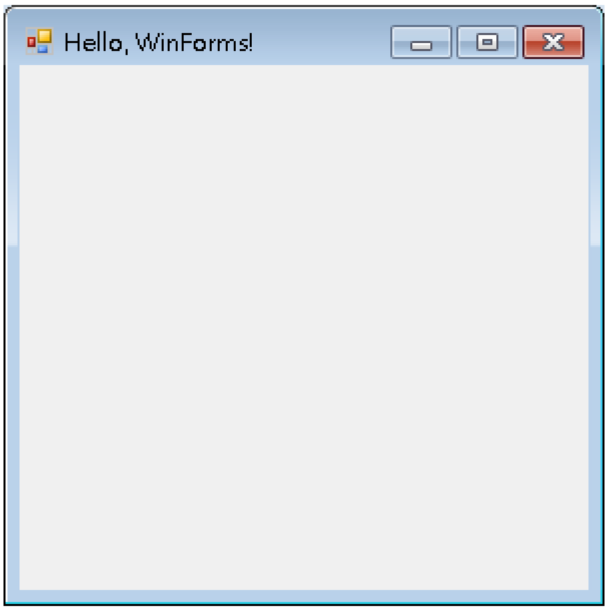


Figure 5 A Hello Form

```
class Program
{
    static void Main(string[] args)
    {
        Form form = new MyFirstForm();
        Application.Run(form);
    }
}

class MyFirstForm : Form
{
    public MyFirstForm()
    {
        Text = "Hello, WinForms!";
        Button button = new Button();
        button.Text = "Click Me!";
        button.Click += ButtonClicked; // A
        Controls.Add(button); // B
    }

    void ButtonClicked(object sender, EventArgs e)
    {
        MessageBox.Show("That's a strong click you've got...");
    }
}
```

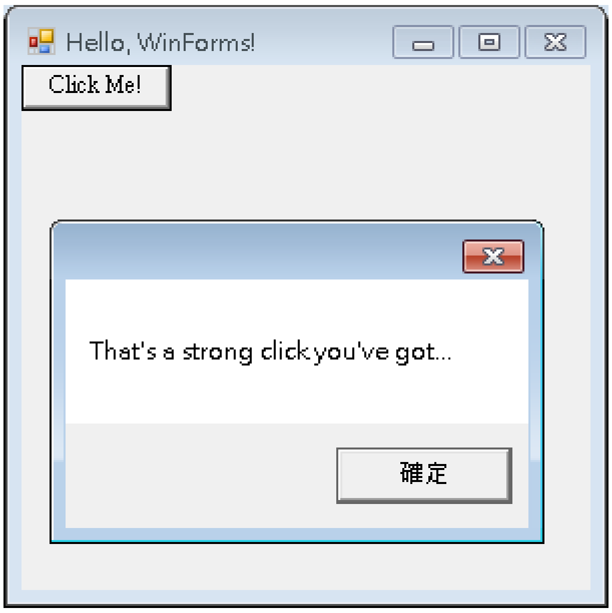


Figure 6 A meesage box invoked by a button

Windows Form Designer

Step 9 Add a Windows Form class

Until now, everything is created by coding in the Main() method, and the Visual Studio does nothing for the developer. However, the Visual Studio provides an easier way to create Windows Form applications. Right click on the "Tutorial1" item, and choose "Windows Form" in the context menu (Figure 7). In the dialog, it doesn't need any change; just click OK. The Visual Studio will create a new class named "Form1" in the project, and switch to Windows Form Designer (Figure 8) directly.

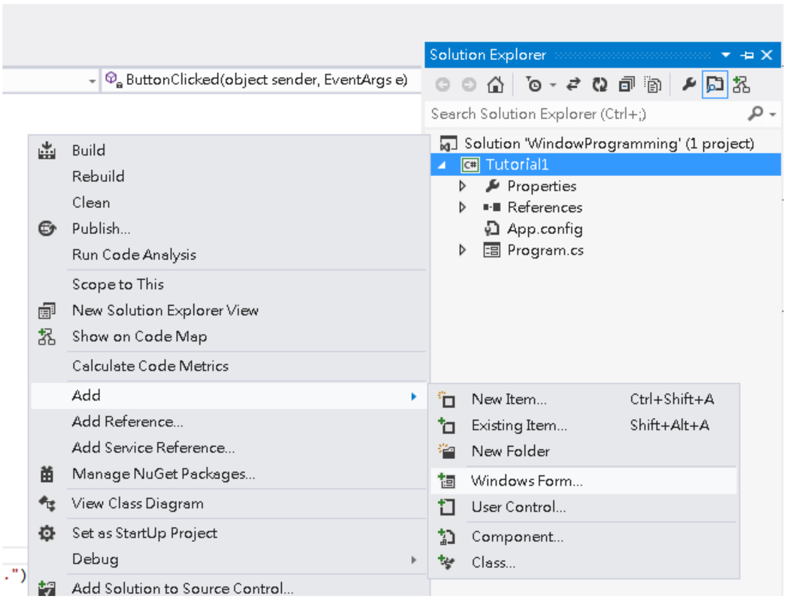


Figure 7 Add a Windows Form class into the project

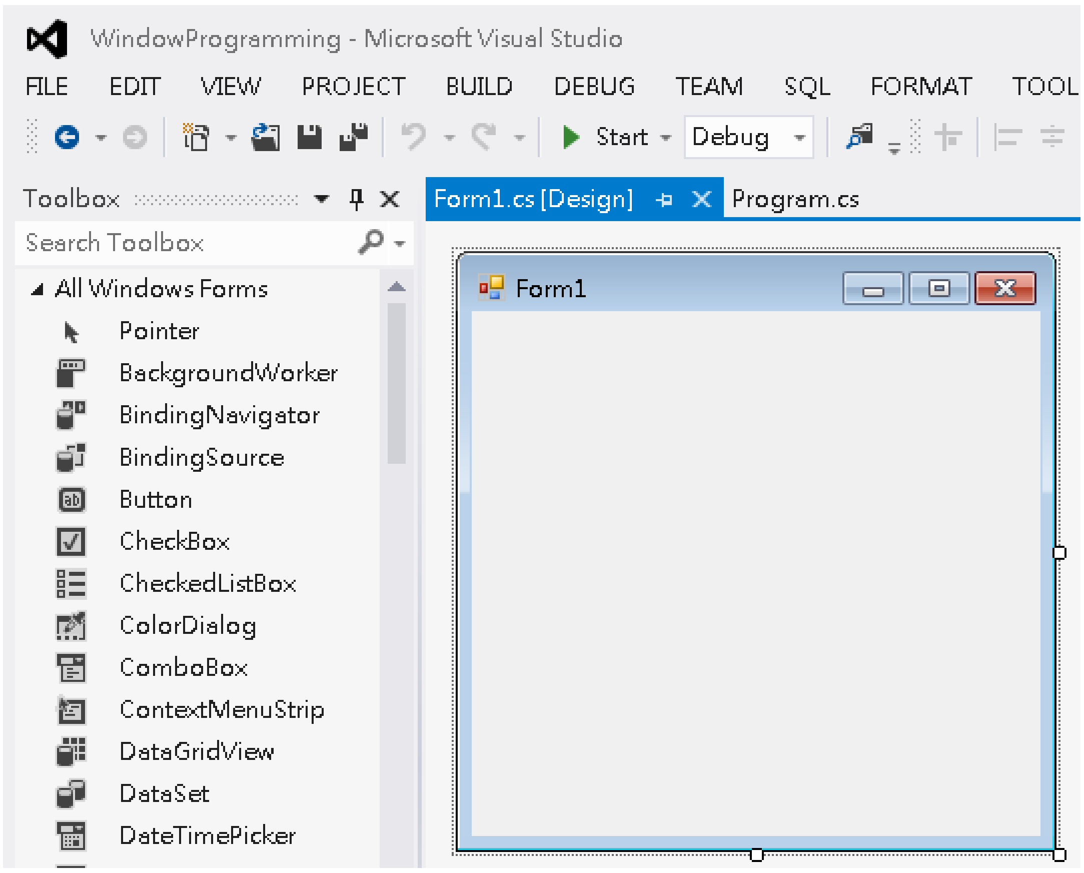


Figure 8 Windows From Designer

Step 10 Layout the Controls

It is not easy to layout lots of controls, but in some case, there is a way to layout controls in regular cells: TableLayoutPanel, which can separate the space into several columns and rows. Then, each cell can contain a control. Drag a TableLaayoutPanel item from the toolbox in the left side of the screen, and drop it in the form (Figure 9).

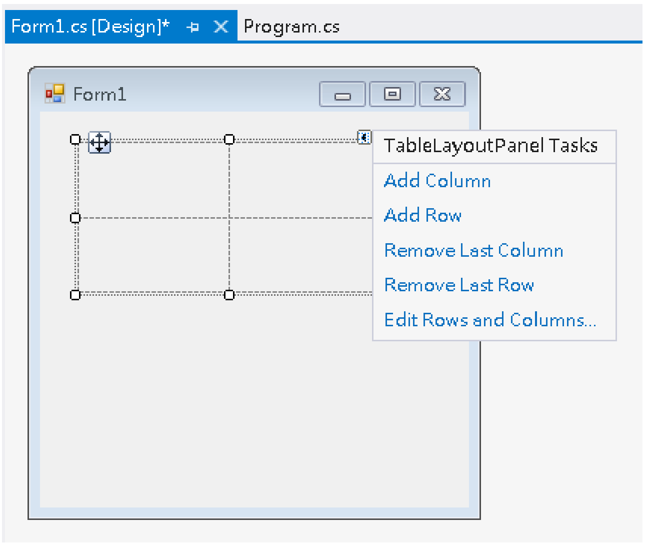
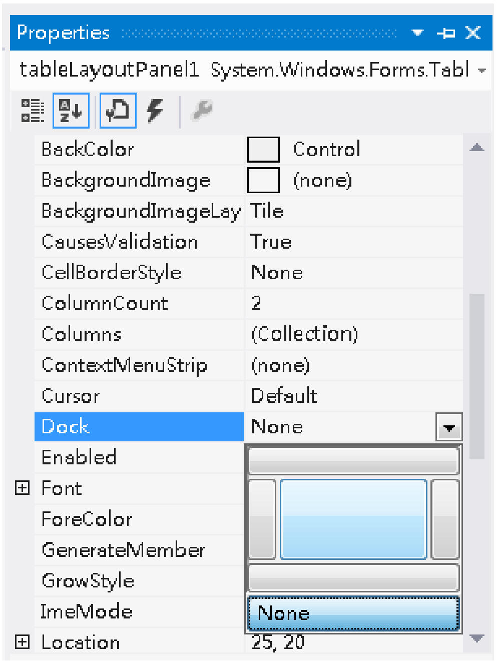


Figure 9 Use the TableLayoutPanel

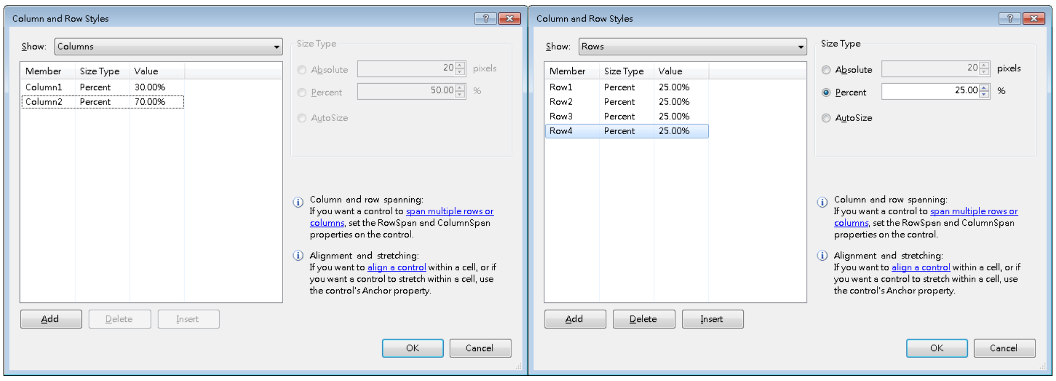
The TableLayoutPanel is added into the form, but the size and its columns and rows are not expected. Add two rows into the table by click "Add Row" item (Figure 10-1)in the menu. Right click the control, and choose the "Properties" item. In the properties panel, find the "Dock" property, and change it from "None" to "Fill." Furthermore, set the height of each row to 25% and the width of the first column to 30% and 70% to the second column in "Edit Rows and Columns" dialog(Figure 10-2).


Properties

tableLayoutPanel1 System.Windows.Forms.Tabl

BackColor	Control
BackgroundImage	(none)
BackgroundImageLay	Tile
CausesValidation	True
CellBorderStyle	None
ColumnCount	2
Columns	(Collection)
ContextMenuStrip	(none)
Cursor	Default
Dock	None
Enabled	
⊞ Font	
ForeColor	
GenerateMember	
GrowStyle	
ImeMode	None
⊞ Location	25, 20

Figure 10-1 Let TableLayoutPanel fill the form


Column and Row Styles

Show: Columns

Member	Size Type	Value
Column1	Percent	30.00%
Column2	Percent	70.00%

Size Type

Absolute 20 pixels

Percent 50.00 %

AutoSize

Column and row spanning:
If you want a control to span multiple rows or columns, set the RowSpan and ColumnSpan properties on the control.

Alignment and stretching:
If you want to align a control within a cell, or if you want a control to stretch within a cell, use the control's Anchor property.

Add Delete Insert

OK Cancel

Column and Row Styles

Show: Rows

Member	Size Type	Value
Row1	Percent	25.00%
Row2	Percent	25.00%
Row3	Percent	25.00%
Row4	Percent	25.00%

Size Type

Absolute 20 pixels

[✓ Percent] 25.00 %

AutoSize

Column and row spanning:
If you want a control to span multiple rows or columns, set the RowSpan and ColumnSpan properties on the control.

Alignment and stretching:
If you want to align a control within a cell, or if you want a control to stretch within a cell, use the control's Anchor property.

Add Delete Insert

OK Cancel

Figure 10-2 Let TableLayoutPanel fill the form

Step 11 Add other Controls

Drag and drop the following controls: Label x 2, Button x 1, and TextBox x 3 into the panel. Modify the Text, TextAlign, and ColumnSpan properties for these controls to let the form looks like the one in Figure 12. Now, in order to let the program use the new designed form, please modify the codes in Main() as following, and then execute the program to see it.

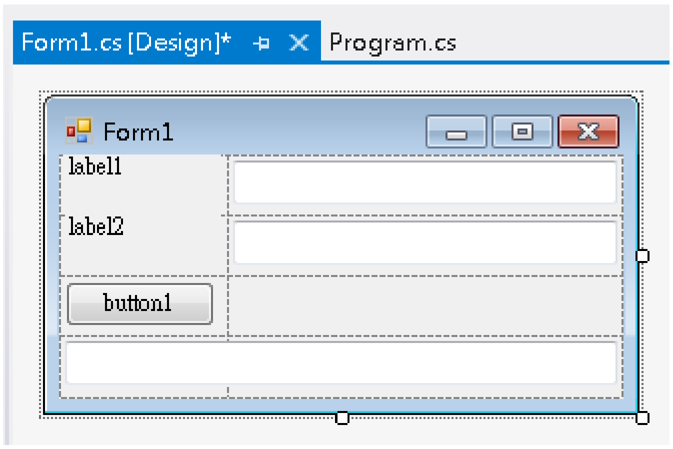
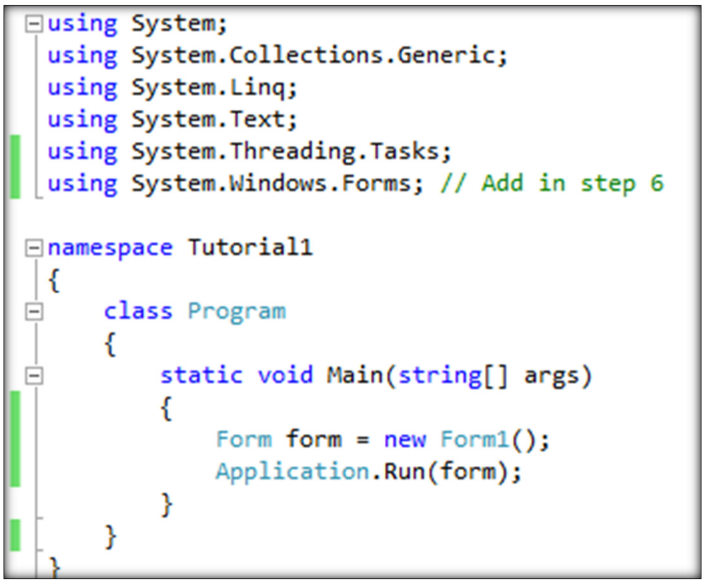


Figure 112 Layout the controls using TableLayoutPanel



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms; // Add in step 6

namespace Tutorial1
{
    class Program
    {
        static void Main(string[] args)
        {
            Form form = new Form1();
            Application.Run(form);
        }
    }
}
```

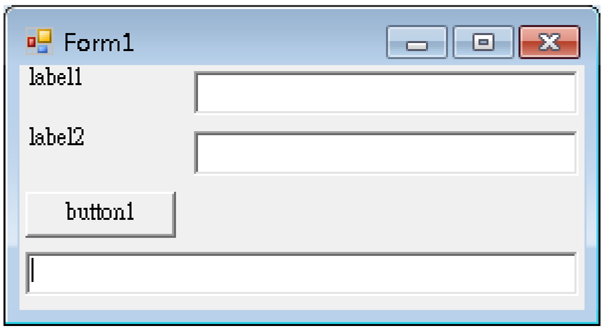
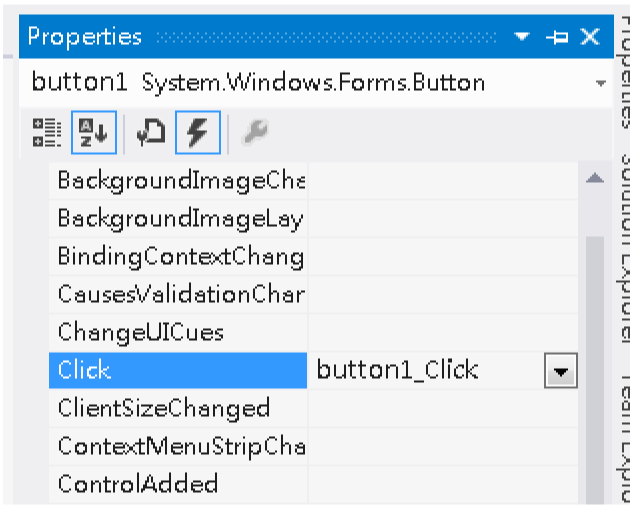


Figure 12 The new form

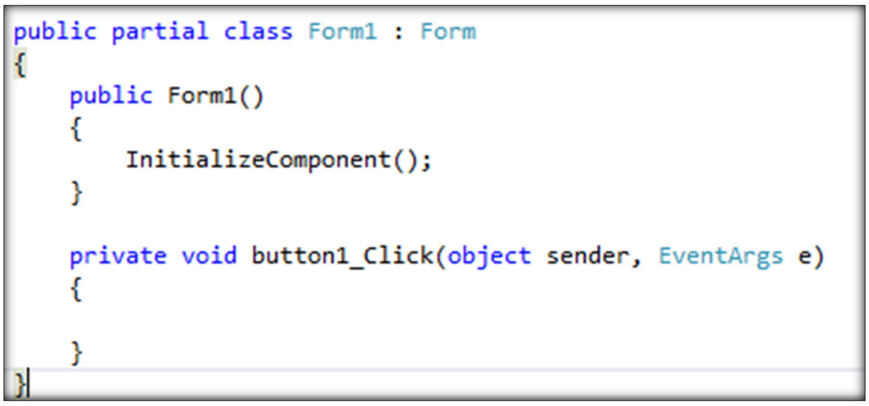
Step 12 It’s time to say “Hello”

The same as the Step 8, it’s time to say “Hello” using the button. Select the “Say Hello” button and click the “Events” button in the Properties panel. Double click on the “Click” item. A fragment of codes is generated by the Visual Studio.



Click: button1_Click

Figure 13 Set the Click event handler in Properties panel

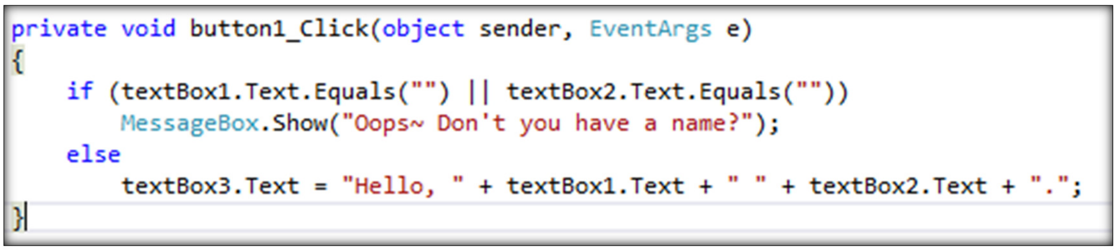


```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {

    }
}
```

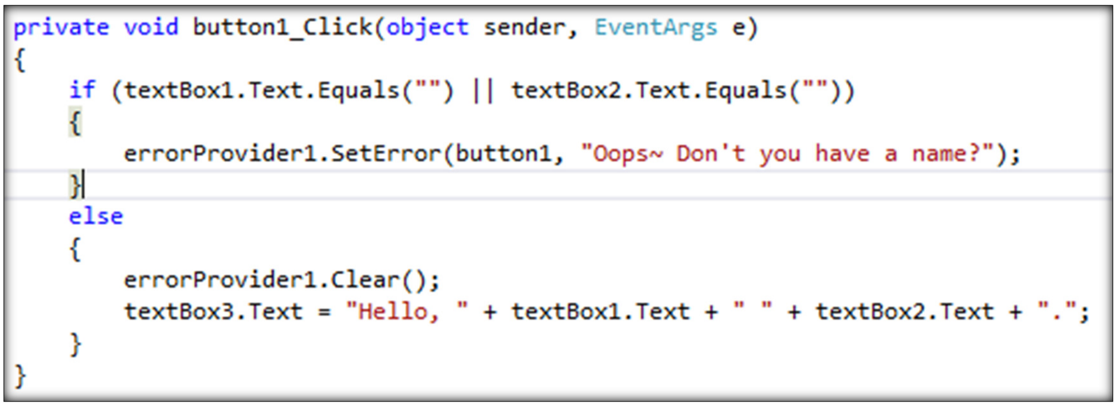
It is impossible for Visual Studio to know what the developer want, so the code can be generated is just a method declaration. Fill the method by the following codes to say “Hello.”



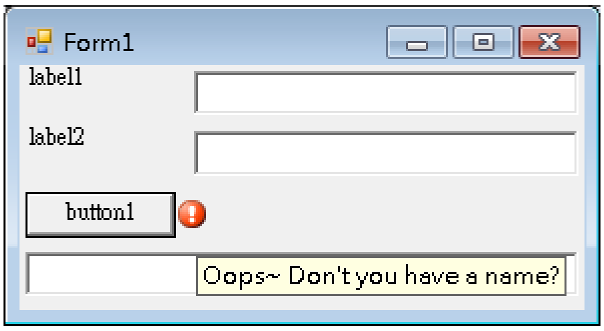
```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Equals("") || textBox2.Text.Equals(""))
        MessageBox.Show("Oops~ Don't you have a name?");
    else
        textBox3.Text = "Hello, " + textBox1.Text + " " + textBox2.Text + ".";
}
```

Step 13 Error Provider

Most people don't like the message box used in the Step 12, so there is another way to show errors. Drag and drop an ErrorProvider into the form. Because the ErrorProvider is not visible controls in normal case, it doesn't appear in the form, but in the bottom side of the designer. Well, modify the codes in button1_Click() method as following. Execute the program, and click "Say Hello" button without input the names to see the error provider how to work (Step 13).



```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Equals("") || textBox2.Text.Equals(""))
    {
        errorProvider1.SetError(button1, "Oops~ Don't you have a name?");
    }
    else
    {
        errorProvider1.Clear();
        textBox3.Text = "Hello, " + textBox1.Text + " " + textBox2.Text + ".";
    }
}
```



Oops~ Don't you have a name?

Figure 14 The Error Provider

-- The End --