Module 3 – Writing C# code for Windows forms.

In the previous module, we have learned the basics of creating graphical user interfaces, which are basically windows forms contain some form controls. These forms will not have any functionally without a backhand code associated with them. For example, to perform some operations when a user clicks on a button, we need to write code to implement the click event handler inside the source code file.

In this chapter, we will write our first C# code. For this, please create a new empty C# windows forms project in Visual Studio. You can name your project as *Module3_1.*

1) Switching between Code and Design views

Your project will have an initial form named **Form1** which you can view in Solution Explorer (you can open it from the *View* menu if not visible). We will open the source code file for Form1. To do that, inside the Solution Explorer window, please right click on the Form1.cs item, which will open a pop-up menu. In this menu, click on **View Code** (See Figure 1).

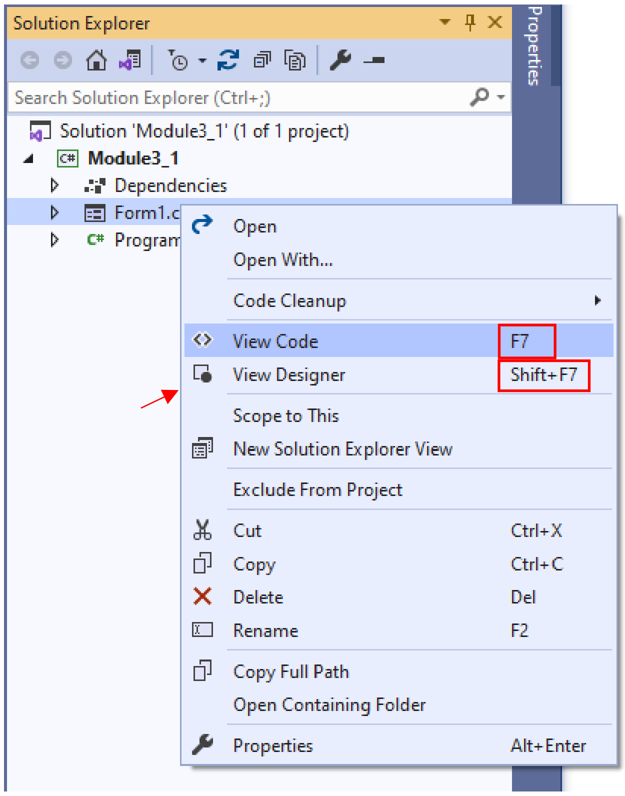
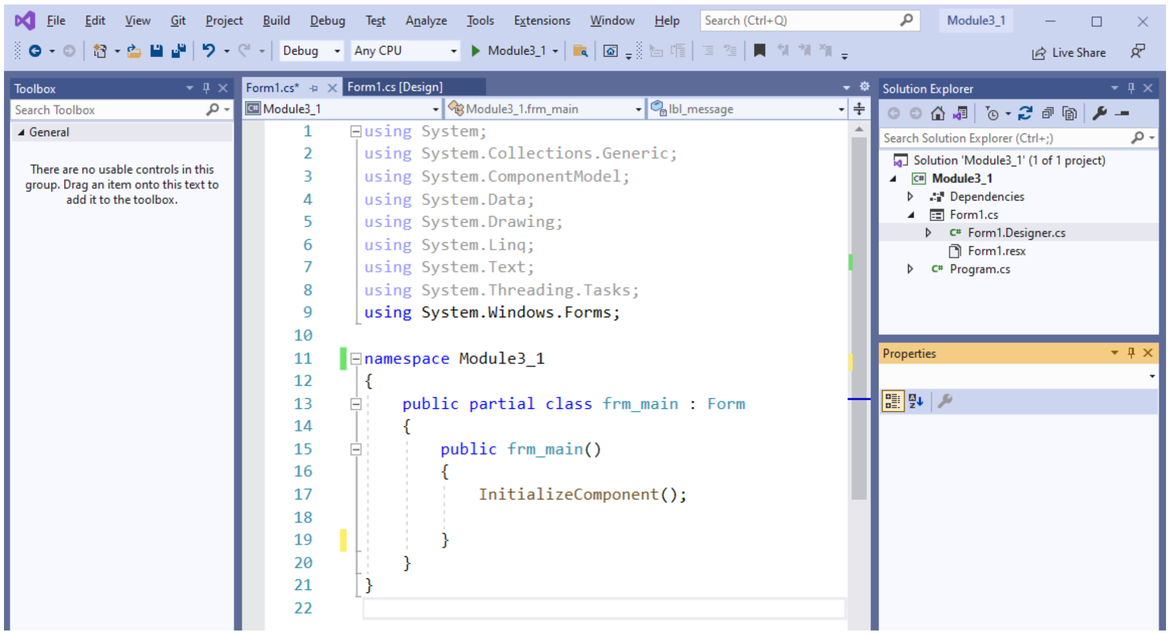


Figure 1. Solution Explorer window.

Alternatively, you can press *F7* to view the code for the selected form (and switch back to the design view by pressing on *Shift+F7*).

The source code file should be opened as shown in Figure 2. You may notice that the Toolbox and Properties windows have no content in the *Code* view. This is because these windows serve for no purpose in coding, and they are mainly intended for designing GUI.



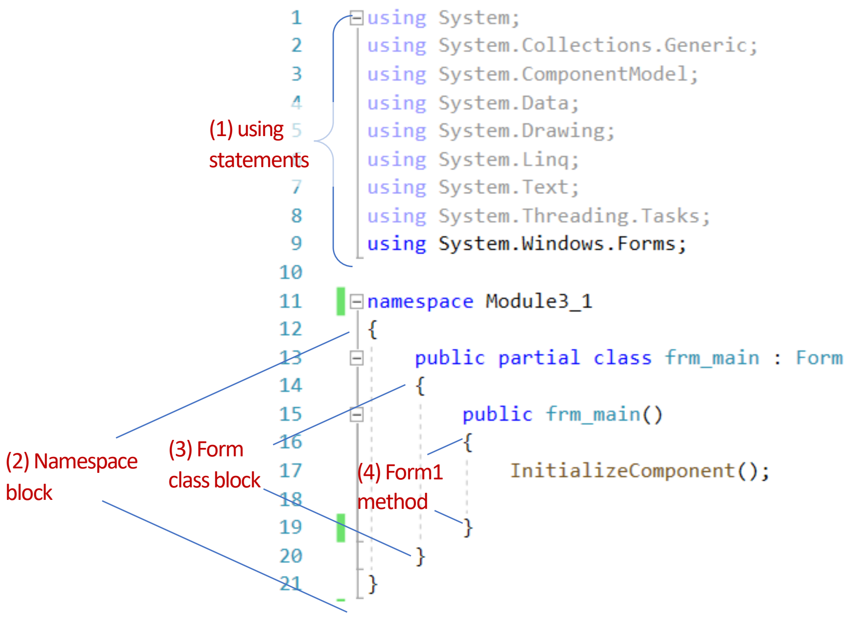
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Module3_1
{
    public partial class frm_main : Form
    {
        public frm_main()
        {
            InitializeComponent();

        }
    }
}

```

Figure 2. Visual Studio environment when code view is enabled.

2) Understanding the structure of the source code file for the form

We will look into the details of the source code file. The C# code is mainly composed of four components: using statements, namespace, class, and methods. These components are marked in Figure 3.

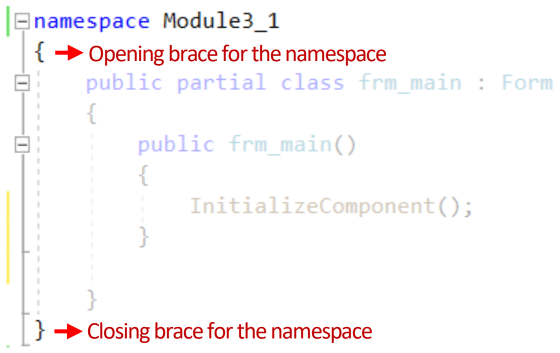


```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Module3_1
{
    public partial class frm_main : Form
    {
        public frm_main()
        {
            InitializeComponent();
        }
    }
}
```

Figure 3. Components of the source code file.

(1) using statements always go to the top of the page and they help import *.Net namespaces* which may contain useful classes that we can take advantage to perform some operations easily and quickly. For examples, importing `System.Text` namespace would allow you to use the `StringBuilder` class, which provides very useful methods to work with strings. By default, popular namespaces are automatically included in the code file for your convenience.

(2) namespace is the overarching container for the classes. Namespace block is automatically created in the code with the name of your project (which is *Week3_1*). Namespaces always have an opening and closing **{ }** braces (see Figure 4). Any code that goes between these braces belongs to the namespace.



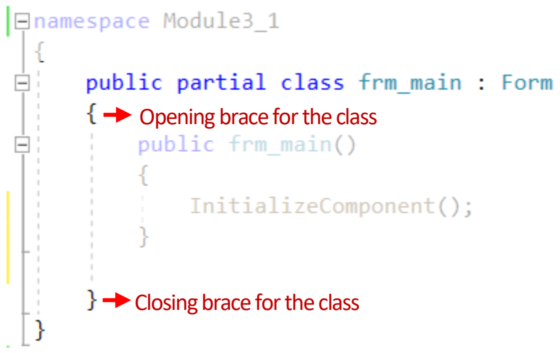
```
namespace Module3_1
{
    public partial class frm_main : Form
    {
        public frm_main()
        {
            InitializeComponent();
        }

    }
}
```

Opening brace for the namespace
Closing brace for the namespace

Figure 4. Opening and closing braces for the namespace.

(3) class serves as the container for the methods that you will write for performing some operations based on user interaction with the interface (e.g., pressing enter, clicking button). C# is an object-oriented programming language, and everything is built based on the classes.

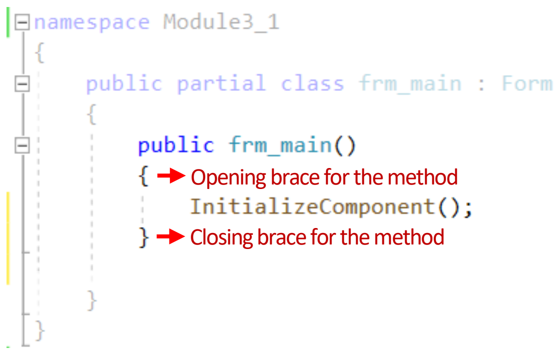
While you can create your custom classes (which will be covered in the following weeks), **frm_main** class (derived from **Form** class) is automatically generated to allow us to write the necessary code to add some functionality to our form. Like namespaces, classes always have an opening and closing braces **{ }** (see Figure 5). Any code that goes between these braces belongs to that class. For now, do discard the **public** and **partial** keywords. These concepts will be covered in the following weeks.



```
namespace Module3_1
{
    public partial class frm_main : Form
    {
        public frm_main()
        {
            InitializeComponent();
        }

    }
}
```

Figure 5. Opening and closing braces for the class.

(4) a default ***constructor*** named **frm_main** is created in the `Form1` class. This constructor contains a system-defined method called `InitializeComponent()`. You should leave it as it is and should never change or remove it. In the next modules, we will learn more about constructors.



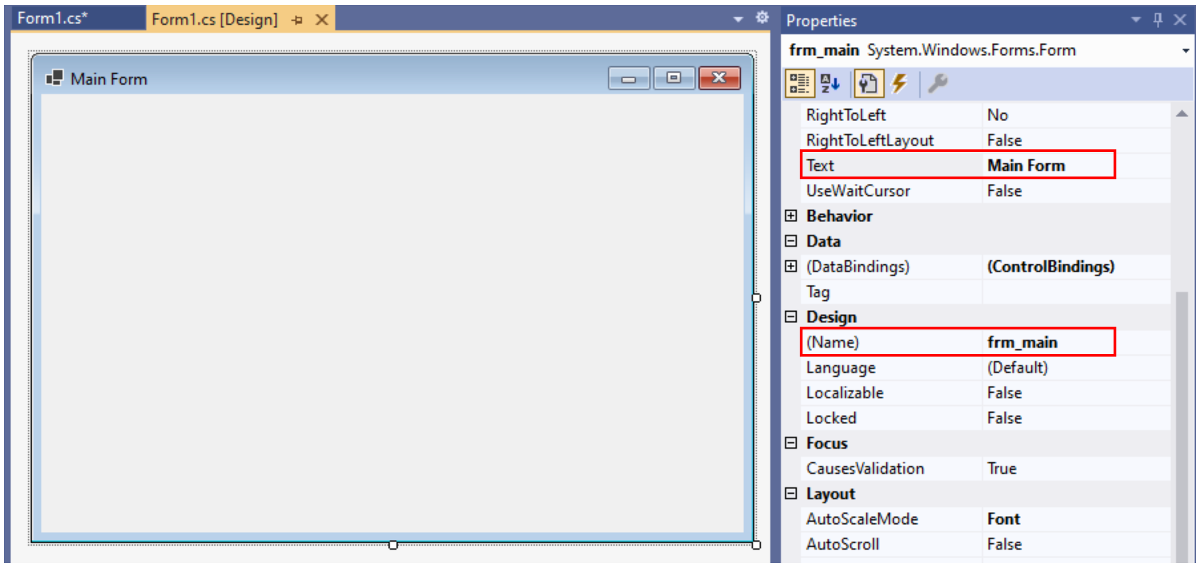
```
namespace Module3_1
{
    public partial class frm_main : Form
    {
        public frm_main()
        { → Opening brace for the method
            InitializeComponent();
        } → Closing brace for the method

    }
}
```

Figure 6. Opening and closing braces for the method.

3) Writing the first C# code to implement a page load event

It is time to get our hands dirty with coding. What we want to achieve is very simple: to display a different text (other than "Initial Message") in the label when the form is loaded for the first time. We will write a single line of code but learn many things. Before starting to code, we will have several preparations in the design view.

If you are in the code view, please press Shift + F7 to switch back to the design view. Click on any part of the form to select it. Then, using the Properties window, set a **Text** (which will be the title of the form) and a **Name** for the form as indicated in Figure 7.

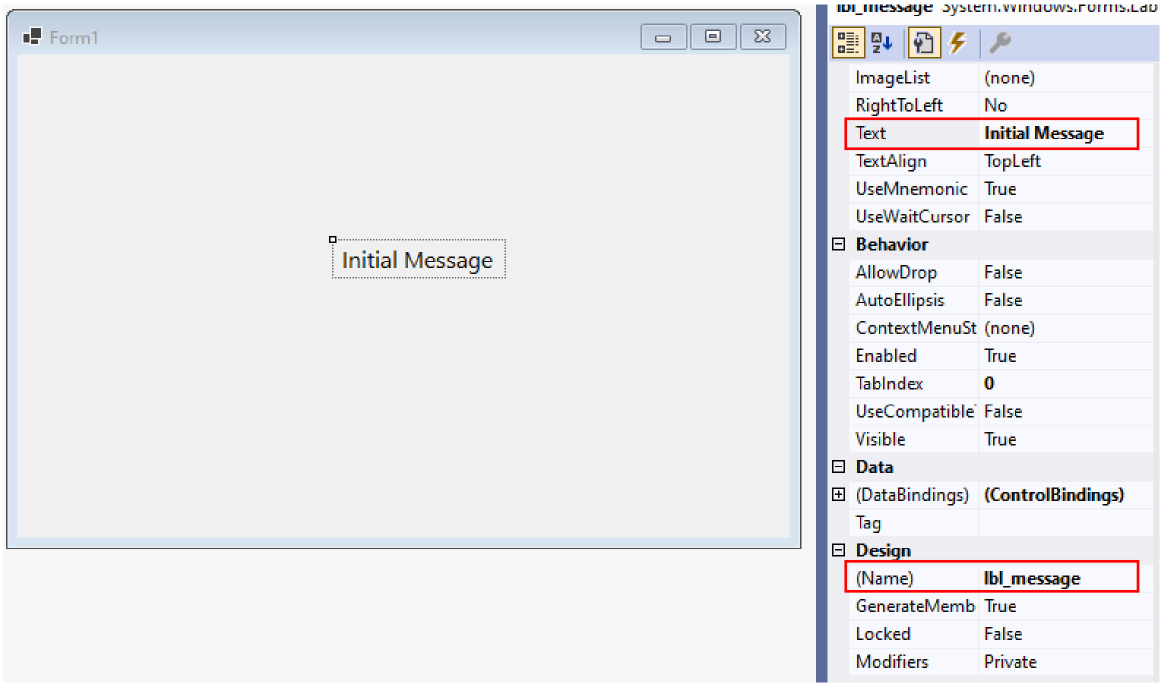


frm_main System.Windows.Forms.Form

RightToLeft	No
RightToLeftLayout	False
Text	Main Form
UseWaitCursor	False
⊞ Behavior	
⊟ Data	
⊞ (DataBindings)	(ControlBindings)
Tag	
⊟ Design	
(Name)	frm_main
Language	(Default)
Localizable	False
Locked	False
⊟ Focus	
CausesValidation	True
⊟ Layout	
AutoScaleMode	Font
AutoScroll	False

Figure 7. Changing the title and the name of the form

Next, add a label control around the centre of the form as seen in Figure 8. Set some initial text (`Initial Message`) for the label and give it a proper name (`lbl_message`).

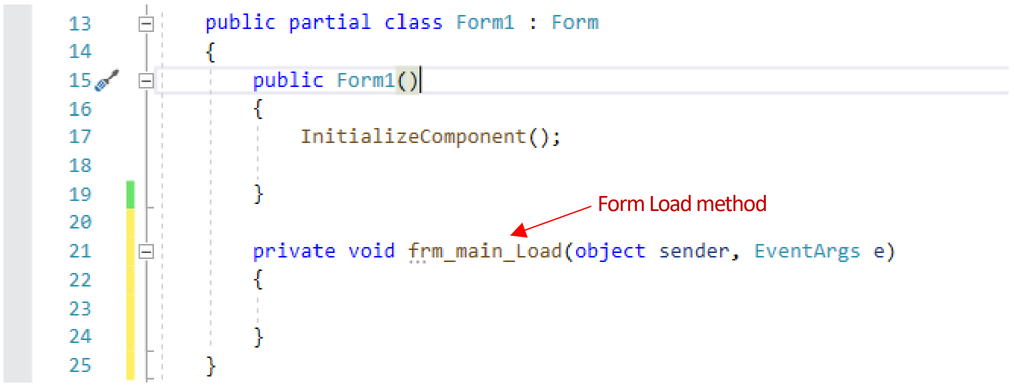


ImageList	(none)
RightToLeft	No
Text	Initial Message
TextAlign	TopLeft
UseMnemonic	True
UseWaitCursor	False
Behavior	
AllowDrop	False
AutoEllipsis	False
ContextMenuSt	(none)
Enabled	True
TabIndex	0
UseCompatible	False
Visible	True
Data	
(DataBindings)	(ControlBindings)
Tag	
Design	
(Name)	lbl_message
GenerateMemb	True
Locked	False
Modifiers	Private

Figure 8. Adding a label control and setting a name and text value.

Now we are ready to start coding. Our goal is to change the text of the label when the page is loaded. C# provides *form loading* method that is triggered when the form is created and loaded in runtime for the first time. Any code written inside this method will be executed when the form is loaded after running the application. To access this method, we will double click on any part of the form (where there is no control). Double clicking on a control (such as label in our case) will generate a different method. So, be careful!

Double click on the form. This will generate the form load method inside the source code file. An automatic name was generated for this method following this structure: *{form name}*_**Load**. Since the form name in our case is *frm_main*, the form load method was set to **frm_main_Load** as seen in Figure 9.

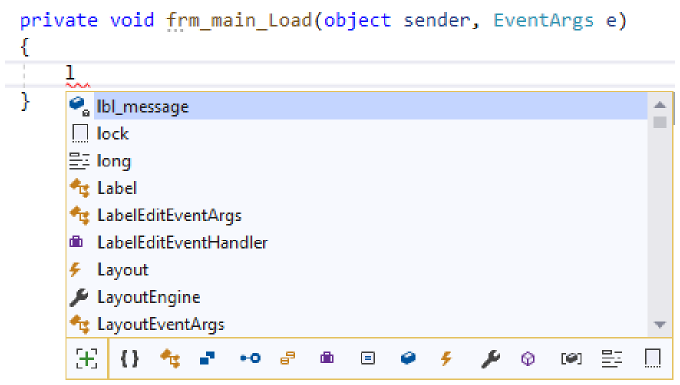


```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

    }

    private void frm_main_Load(object sender, EventArgs e)
    {

    }
}
```

Figure 9. Source code file with Form Load method generated.

Whatever code we write here will be executed when the form is created and loaded for the first time in runtime. What we want to do is to change the text of the label (`lbl_message`) during form load. To reference a control in the code view, you need to type its name (in the correct line in the source code). Before writing the code, click inside the **`frm_main_Load`** method to place the curser inside this method.

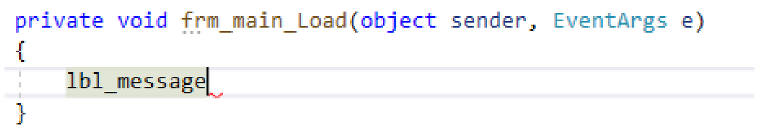
Now you are ready to write your first code. Start typing the name of the label, which is `lbl_message`. When you type only the letter *l*, a popup window will appear to provide some smart suggestions about what you might be searching for (see Figure 10). The first item in the list (which is already selected as indicated with blue background) is what we are looking for.



```
private void frm_main_Load(object sender, EventArgs e)
{
    l
}
```

Figure 10. Smart suggestions for coding.

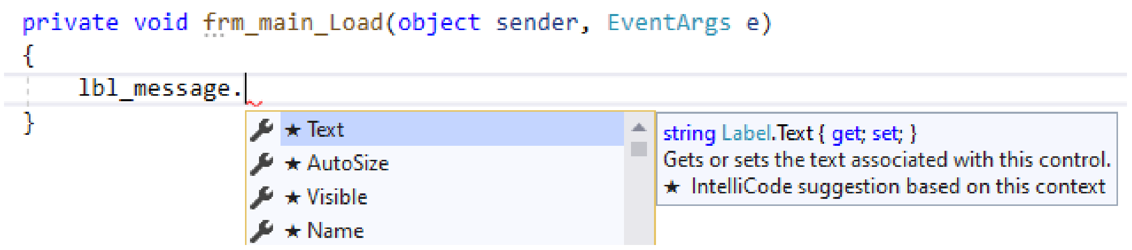
Please press *Enter* or *Tab* key to quickly add `lbl_message` as seen in Figure 11 below.



```
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message
}
```

Figure 11. lbl_message is added to the code.

These smart suggestions are part of the **IntelliSense** technology available in Visual Studio. IntelliSense anticipates what you want to type and helps you speed up the coding process. IntelliSense has many more features and you will notice them as you continue practice writing code. For example, if you add a period (.) just after lbl_message, a new window will popup which includes a list of items associated with the label control. See Figure 12 for the illustration.

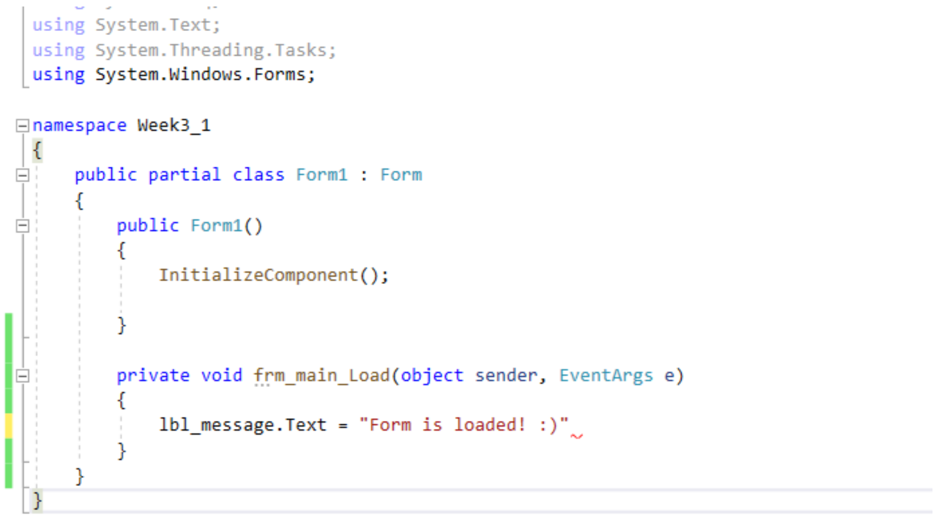


```
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message.
}
```

string Label.Text { get; set; }
Gets or sets the text associated with this control.
★ IntelliCode suggestion based on this context

Figure 12. Intelligence popup list after typing (.)

We choose the Text item in the list and continue writing our code. We need to change the label text, in other words, we want to *assign* a new string to the **Text** property of the label. To assign a new value, we use the assignment operator (=). Since the Text property accepts only strings, we provide a string value inside quotation marks: “`Form is loaded! :)`”, as seen in Figure 13.



```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Week3_1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

        }

        private void frm_main_Load(object sender, EventArgs e)
        {
            lbl_message.Text = "Form is loaded! :)"
        }
    }
}
```

Figure 13. The code for assigning a string to a label text.

Now, we are ready to run and test our application. To do that we have two options under the **Debug** menu: Start Debugging [F5] or Start Without Debugging [Ctrl+F5] (see Figure 14).

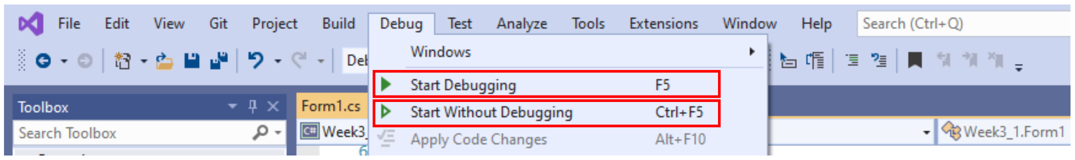


Figure 14. Debugging options in the Debug menu.

Start Debugging [F5] option is also available in the toolbar as shown in Figure 15.

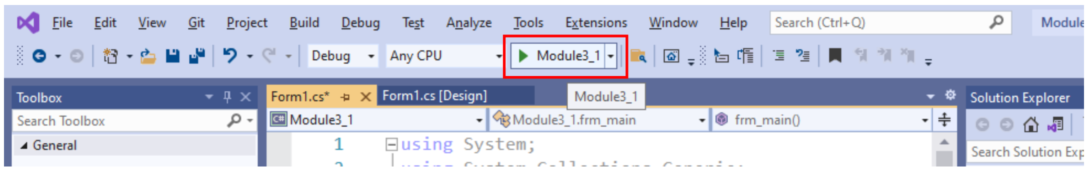
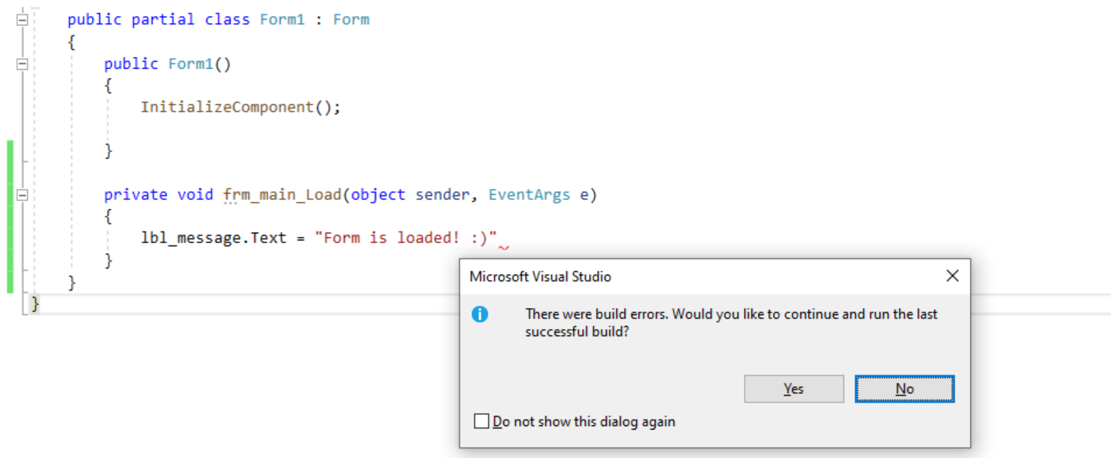


Figure 15. Debugging option in the toolbar.

If you run your application with debugging, you can add breakpoints to track what is happening in your code line by line. For now, we do not need the debugging option since we have a simple code. Please press Ctrl+F5 in your keyboard to run your application without debugging. You should receive a warning about the errors in your code as you can see Figure 16. We have some errors in our code, and Visual Studio warn us about them before running our application.



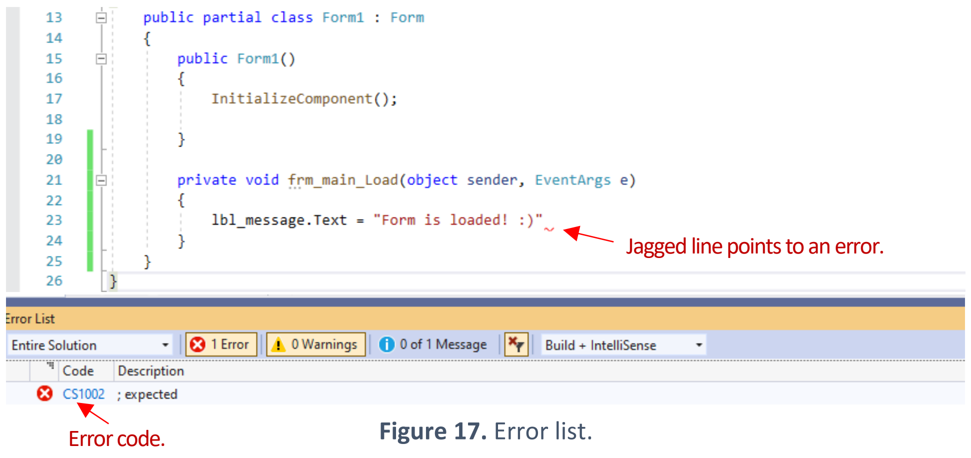
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

    }

    private void frm_main_Load(object sender, EventArgs e)
    {
        lbl_message.Text = "Form is loaded! :)"
    }
}
}
```

Microsoft Visual Studio

There were build errors. Would you like to continue and run the last successful build?

Figure 16. Debugging option in the toolbar.

Please click NO, which should open the **Error List** window on the bottom of the code window as seen in Figure 17. The error message says "*; expected*", which is actually highlighted in the code for your convenience with a **jagged red line**.

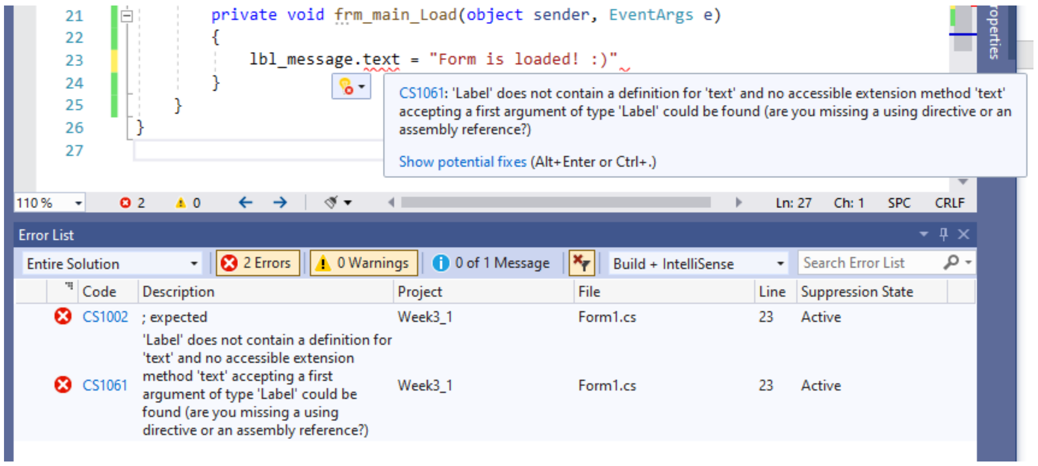


```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

    }

    private void frm_main_Load(object sender, EventArgs e)
    {
        lbl_message.Text = "Form is loaded! :)"
    }
}
}
```

CS1002 ; expected

Figure 17. Error list.

As you write code, Visual Studio analyses your code instantly to identify the syntax errors (like grammar errors in a spoken language). As an example, please change .*Text* with .text in your code to intentionally throw an error. There should appear a jagged line just under the .text as you can see in Figure 18 below. While these errors are listed in the Error List window, you can also mouse over on each jagged line to get more information about each error.



```
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message.text = "Form is loaded! :)"
}
}
```

CS1061: 'Label' does not contain a definition for 'text' and no accessible extension method 'text' accepting a first argument of type 'Label' could be found (are you missing a using directive or an assembly reference?)

Show potential fixes (Alt+Enter or Ctrl+.)

110 % | 2 | 0 | Ln: 27 | Ch: 1 | SPC | CRLF

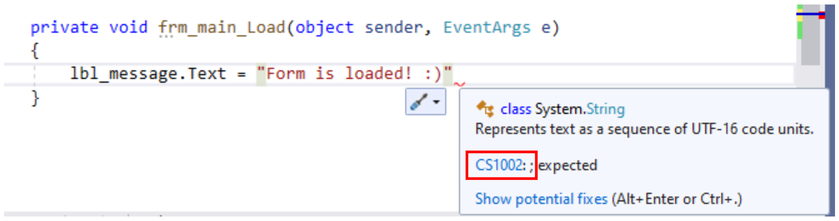
Error List

Entire Solution | 2 Errors | 0 Warnings | 0 of 1 Message | Build + IntelliSense | Search Error List

	Code	Description	Project	File	Line	Suppression State
✖	CS1002	; expected	Week3_1	Form1.cs	23	Active
✖	CS1061	'Label' does not contain a definition for 'text' and no accessible extension method 'text' accepting a first argument of type 'Label' could be found (are you missing a using directive or an assembly reference?)	Week3_1	Form1.cs	23	Active

Figure 18. Error messages in the application.

Please fix your code by changing .text back to .Text. Let's focus on the first error we received. Mouse over the jagged line to see the error message.



```
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message.Text = "Form is loaded! :)"
}
```

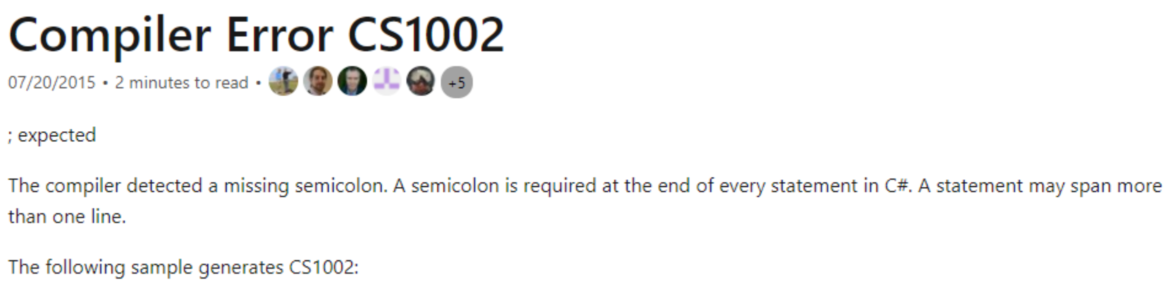
class System.String
Represents text as a sequence of UTF-16 code units.

CS1002: ; expected

Show potential fixes (Alt+Enter or Ctrl+.)

Figure 19. Error message is shown in the popup window.

The error tells us that we need to add a **semicolon (;)** somewhere in the code. To get more information about the error, you can click on the Error code (CS1002) in the error window, which should open the following webpage in your browser (see Figure 20).



Compiler Error CS1002

07/20/2015 • 2 minutes to read • +5

; expected

The compiler detected a missing semicolon. A semicolon is required at the end of every statement in C#. A statement may span more than one line.

The following sample generates CS1002:

Figure 20. Detailed information about the error.

In C#, the termination of a code statement in a line is indicated via semicolon. At the end of each code line, we have to add a semicolon. Please add a semicolon to fix the error as indicated in Figure 21 and run your application again.

```
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message.Text = "Form is loaded! :)";
}
```

Figure 21. Final code to change the label text.

The form should be loaded with the correct message displayed in the label (see Figure 22).

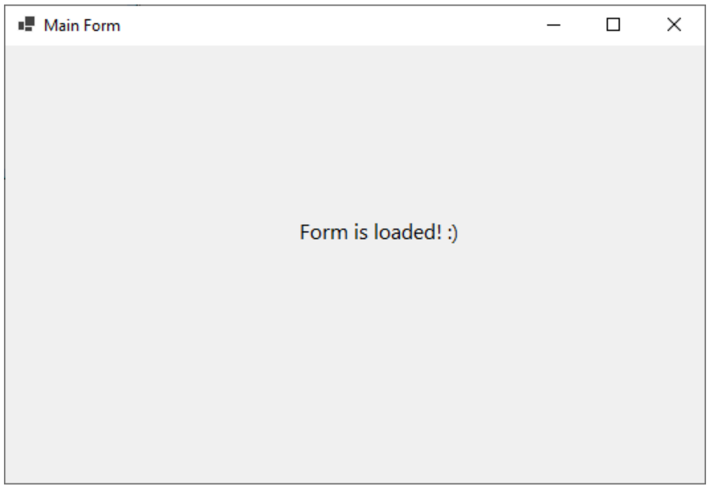


Figure 22. Form after running the application.

4) Implementing button click events

Clicking a button is a very typical and common user action performed in forms. We will add two buttons to our form. These buttons will serve for changing the label text when clicked by the user. First, please add two buttons under the label and change their Text and Name properties as suggested in Figure 23.

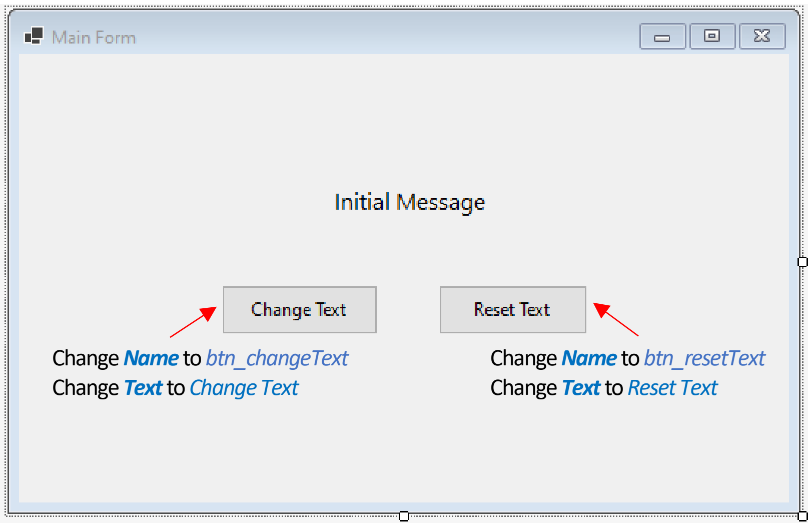
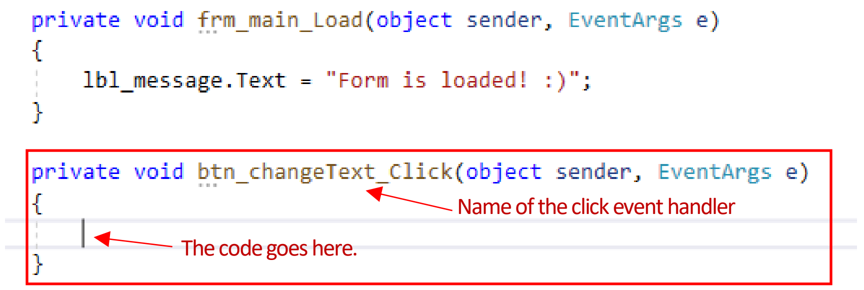

Main Form
Initial Message
Change Text
Reset Text
Change **Name** to *btn_changeText*
Change **Text** to *Change Text*
Change **Name** to *btn_resetText*
Change **Text** to *Reset Text*

Figure 23. Adding two buttons to the form.

Our goal is to change the label text to "*Text has been changed*." when the first button (`btn_changeText`) is clicked, and to change the label text to "*Initial Message*" when the second button (`btn_resetText`) is clicked.

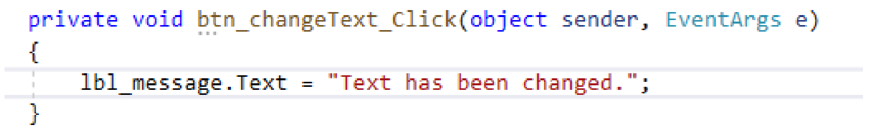
All code that we want to execute when a button is clicked should go inside that button's click event handler. That means we need to first create a click event handler for each button separately. To do that, please first double-click on the `btn_changeText` button, which should open the source code file with the click event handler created (just under the `frm_main_Load` form load handler), as seen in Figure 24 below.



```
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message.Text = "Form is loaded! :)";
}

private void btn_changeText_Click(object sender, EventArgs e)
{
    |
}
```

Figure 24. Click event handler for btn_changeText.

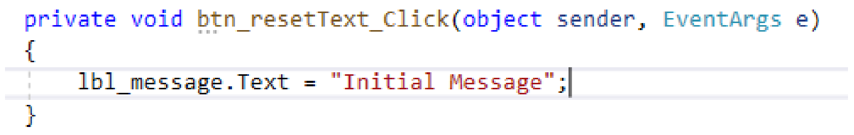
The name of the click event handler, `btn_changeText_Click`, is automatically created by appending the `_Click` keyword to the end of the button name (`btn_changeText`). Any code we want to execute should go between the opening and closing braces `{ }`. We will add the code to change the text of the label (as we did for the form load event handler). See Figure 25.



```
private void btn_changeText_Click(object sender, EventArgs e)
{
    lbl_message.Text = "Text has been changed.";
}
```

Figure 25. Adding code inside `btn_changeText_Click` to change the label text.

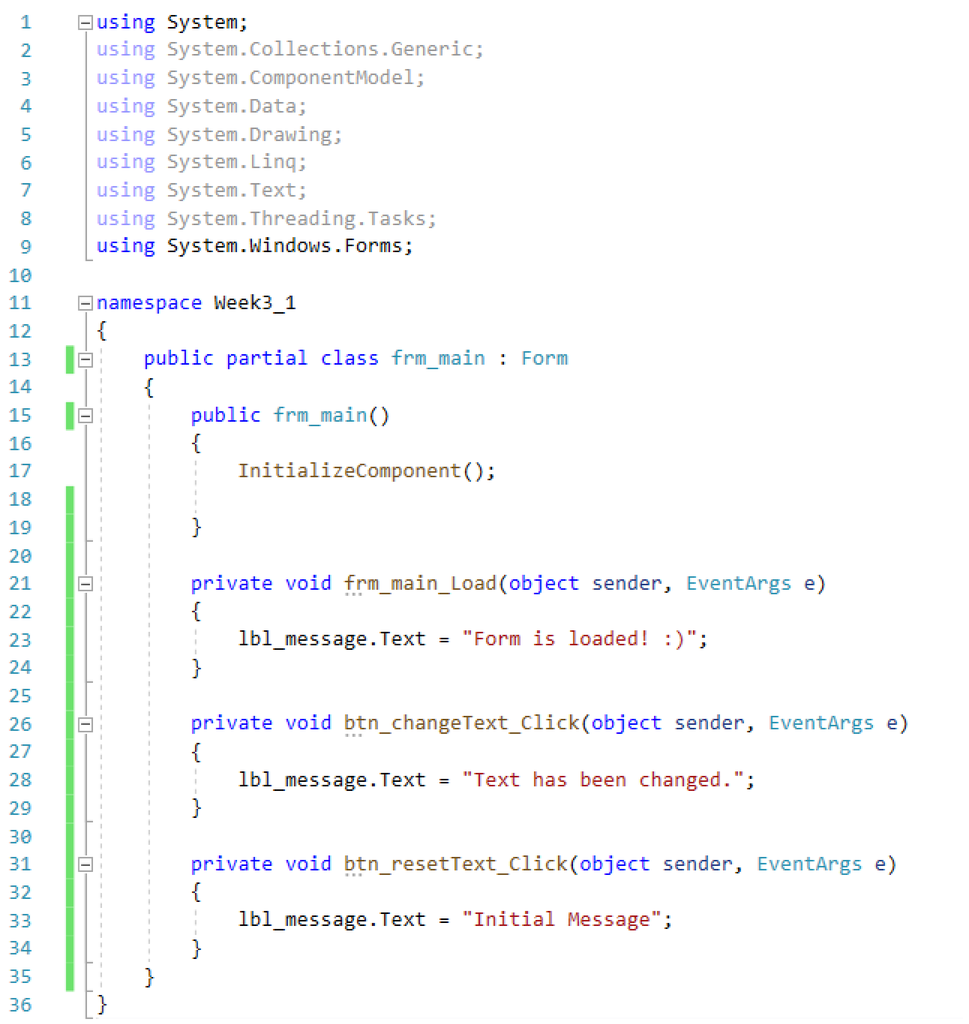
We will repeat the same process for the `btn_resetText` button. Please press *Shift+F7* to switch to the design view, and then double click on the `btn_resetText` button to add a click event handler. Then write the necessary code to change the label text as intended. Figure 26 shows the implementation of the `btn_resetText_Click` click event handler.



```
private void btn_resetText_Click(object sender, EventArgs e)
{
    lbl_message.Text = "Initial Message";
}
```

Figure 26. Adding code inside `btn_resetText_Click` to change the label text.

The content of the complete source code file is shown in Figure 27.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Week3_1
{
    public partial class frm_main : Form
    {
        public frm_main()
        {
            InitializeComponent();

        }

        private void frm_main_Load(object sender, EventArgs e)
        {
            lbl_message.Text = "Form is loaded! :)";
        }

        private void btn_changeText_Click(object sender, EventArgs e)
        {
            lbl_message.Text = "Text has been changed.";
        }

        private void btn_resetText_Click(object sender, EventArgs e)
        {
            lbl_message.Text = "Initial Message";
        }
    }
}
```

Figure 27. The complete source code file.

Please press Control+F5 key combination to test your application. First, “*Form is loaded:)*” text should be displayed as seen in Figure 28.

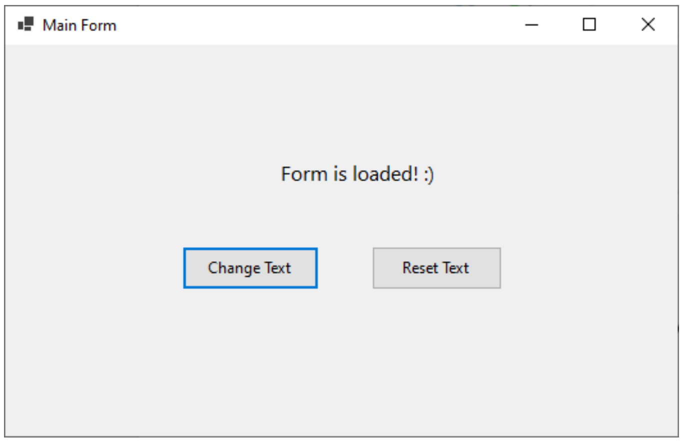


Figure 28. The form is loaded for the first time.

Then, please click the *Change Text* button on the left. The label text should change to "*Text has been changed*." as shown in Figure 29 below.

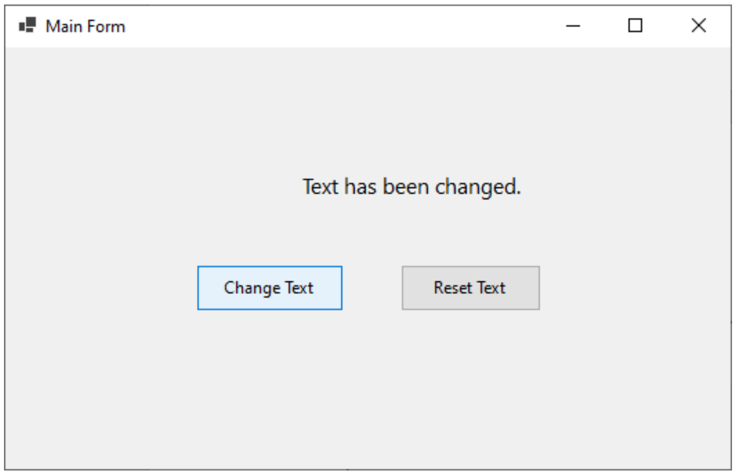


Figure 29. Form view after the Change Text button is clicked.

Now, please click on *Reset Text* button. The label text should change to "*Initial Message*" as you can see in Figure 30.

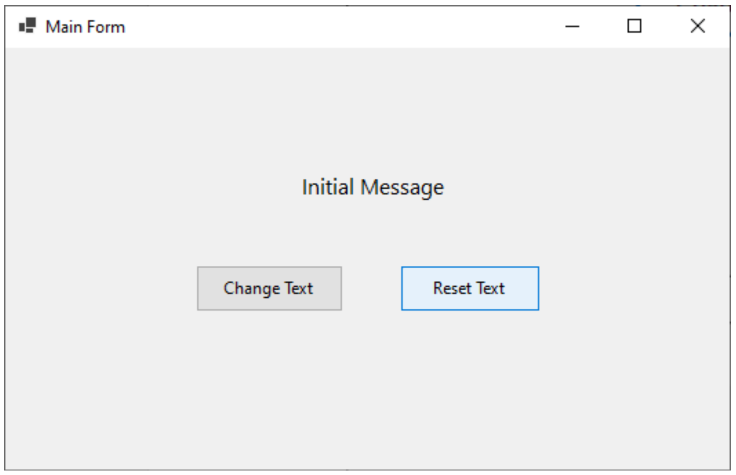
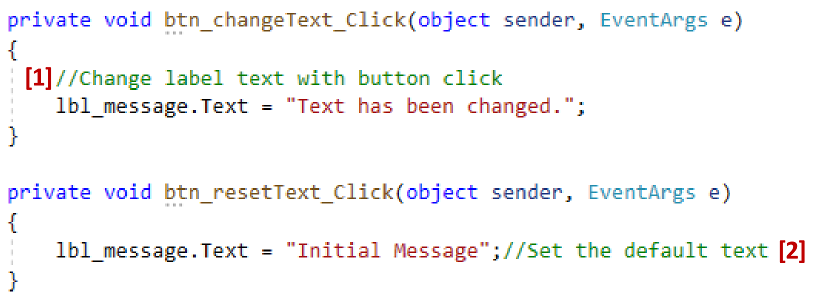


Figure 30. Form view after the Reset Text button is clicked.

4) Adding comments to explain your code

A very important practice in coding is to add comments in your code to explain your code and the programming logic. These comments are quite necessary in particular when you write long and complex code in more comprehensive projects. It is very normal that you forget about code that you wrote a week or month ago. If you have added proper explanations within the code, they will remind you about what you were tyring to do with each specific line or block of code. These comments are of more importance if you work in collaborative projects where your peers need to make sense of your code with minimum effort. In runtime, the compiler ignores the comments since they are not part of your programming logic but just some descriptive text.

In C# you can add two main types of comments. First one is line comments, which starts with two forward slashes (`//`). After `//`, whatever you type in the same line is a comment and marked with green colour. Figure 31 shows two examples of line comment.

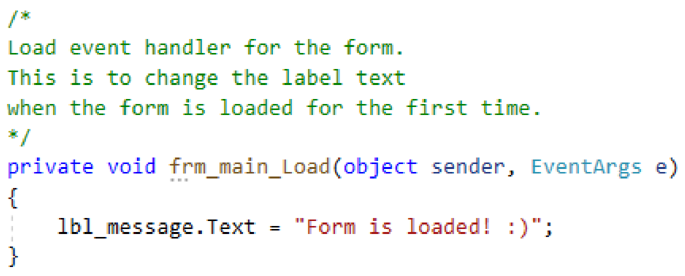


```
private void btn_changeText_Click(object sender, EventArgs e)
{
 [1]//Change label text with button click
    lbl_message.Text = "Text has been changed.";
}

private void btn_resetText_Click(object sender, EventArgs e)
{
    lbl_message.Text = "Initial Message";//Set the default text [2]
}
```

Figure 31. Adding line comments.

As seen in Figure 31, line comments can start in a new line like **[1]** or they can be added to the end of a code line like **[2]**.

Another common type is the block comments, which consist of several lines of comments as its names suggests (although it also can have only one line of comment). The beginning of block comments is marked with `/*` (a forward slash followed by an asterisk) and the end of block comments is marked with `*/` (an asterisk followed by a forward slash). Figure 32 shows an example of block comment.



```
/*
Load event handler for the form.
This is to change the label text
when the form is loaded for the first time.
*/
private void frm_main_Load(object sender, EventArgs e)
{
    lbl_message.Text = "Form is loaded! :)";
}
```

Figure 32. Adding block comments.

5) Using textboxes to obtain user input

Often in our applications we need input from users. These inputs can be also called user data and they can be collected through various form controls (quite similar to the web interfaces), such as textboxes, dropdown lists, checkboxes, and radio buttons. Once the users submit their data, we need to perform some operations on them depending on our goal (e.g., registering a user, computing an average score).

One common way to collect user data is to ask users enter some free text inside a text field. In Windows forms, these text fields are added using a **Textbox** control. Before continuing with the rest of the chapter, please create a new application, called *Module3_2*.

Textbox control is located under the *Common Windows Forms* section of the toolbox. Just to try out, please place a textbox control inside the form as shown in Figure 33 below.

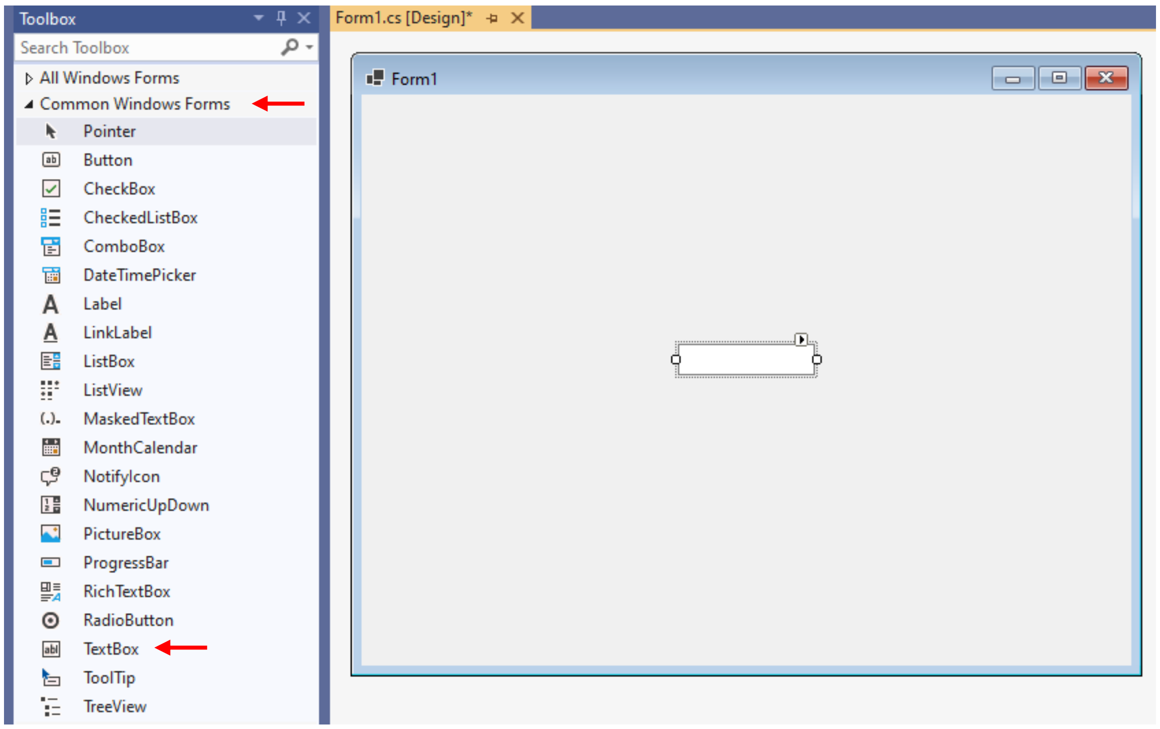


Figure 33. Adding a Textbox control to the form.

In the design view, it is not allowed to type into the textbox. Press Control+F5 to run your application. As shown in Figure 34, now in runtime you should be able to enter some text inside the textbox.

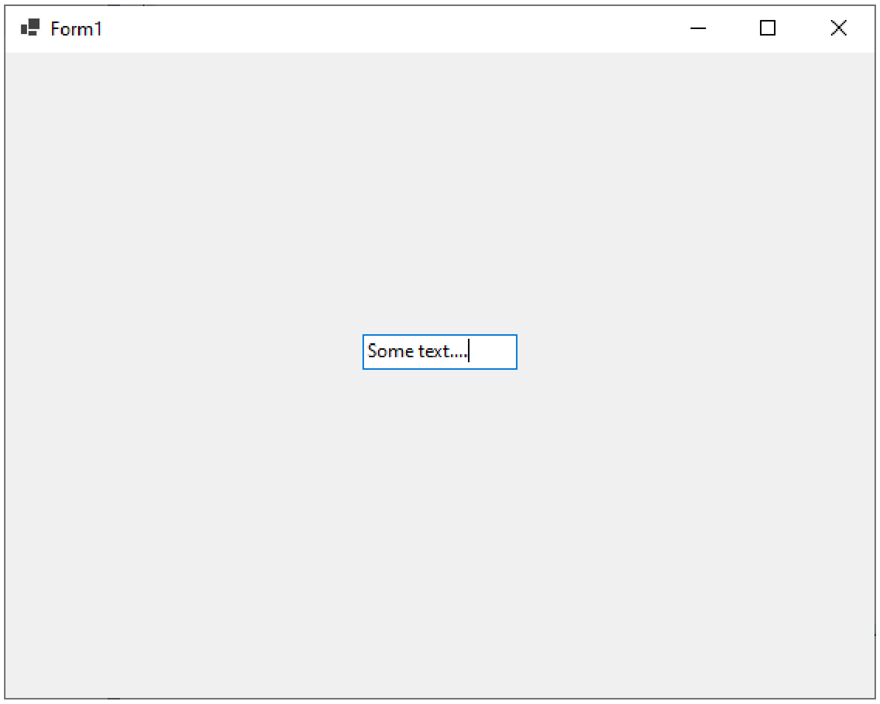
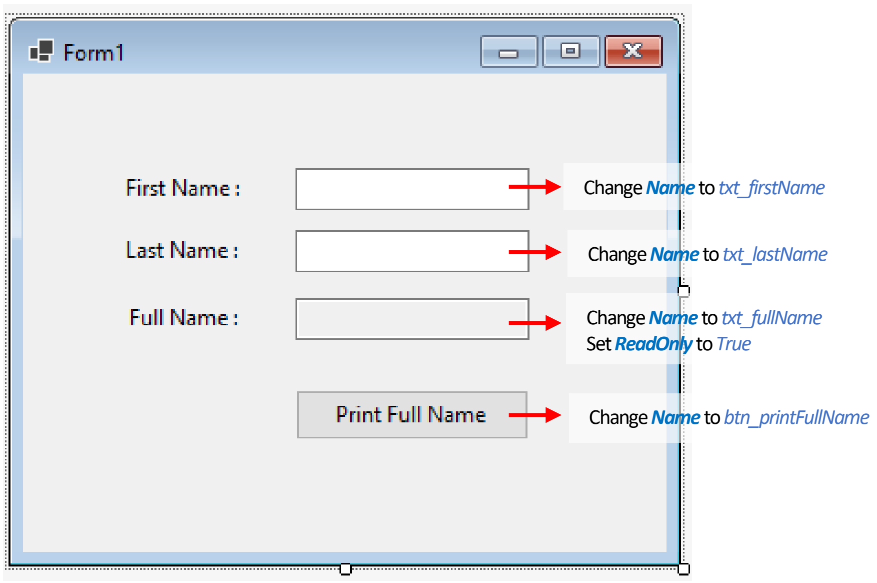


Figure 34. Typing some text inside the textbox in runtime.

But how do we access (from the source code file) the value typed inside the textbox? We use its **Text** property. Anything typed inside the textbox is stored in its Text property.

In this project we will program an application which prints users' full name based on the first name and last name provided. Delete the textbox and add three labels with three adjacent textboxes as seen in Figure 35. Just under them, please add a button control. Update the text and name properties of all controls as suggested in the figure.

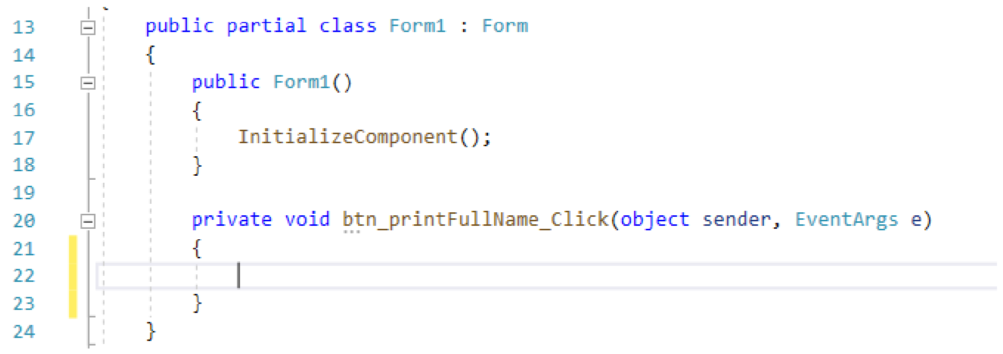


Change *Name* to *txt_firstName*
Change *Name* to *txt_lastName*
Change *Name* to *txt_fullName*
Set *ReadOnly* to *True*
Change *Name* to *btn_printFullName*

Figure 35. Form design of the application to print full name.

You might have noticed that the third textbox, named `txt_fullName`, has a different background colour than the other two textboxes. This is because the **ReadOnly** property of `txt_fullName` is set to `True`. We made this change because `txt_fullName` will only serve for displaying a text and therefore allowing users to change its content is not necessary. The background colour of read-only textboxes is automatically set to grey (although it can be changed).

Our goal is that when `btn_printFullName` is clicked, the text entered into `txt_firstName` and `txt_lastName` should be merged into a single string and displayed in `txt_fullName`. As you may guess, we need to create a click event handler for `btn_printFullName`, and then write all necessary code inside this handler. To automatically create the handler, please double click on `btn_printFullName`. You should obtain the following screen shown in Figure 36.



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

    private void btn_printFullName_Click(object sender, EventArgs e)
    {
        
    }
}
```

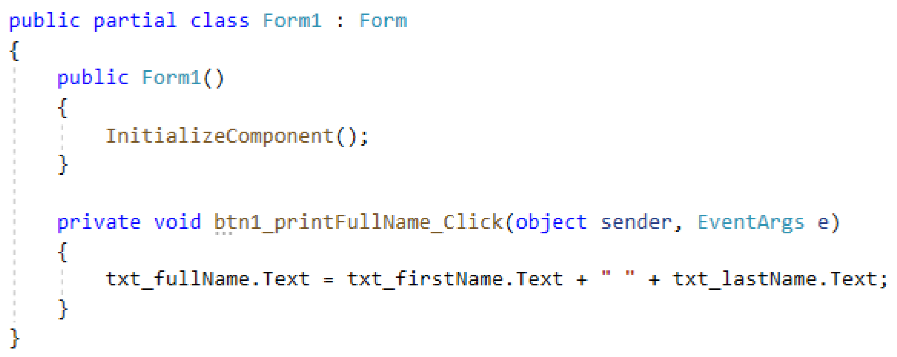
Figure 36. Click event handler for `btn_printFullName`.

Your source code file should have the `using` statements at the top. These are cut off in Figure 36 to save space.

6) Merging the first name and last name

To merge two (or more) strings, we can use the plus (**+**) operator. For example, `"Visual" + " " + "C#"` would print `"Visual C#"`. To merge the first name (typed into `txt_firstName`) and last name (typed into `txt_lastName`), we can also use the **+** operator.

You may remember that we use the **Text** property to obtain the contents of a textbox. We use the same property also for changing the content of a textbox. What we need to do is merge the first name and last name values entered by the user (i.e., `txt_firstName.Text + " " + txt_lastName.Text`) to obtain the full name. Next, we need to assign this merged string to the **Text** property of the `txt_fullName` control so that it can display the full name. Complete code is provided in Figure 37.



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

    private void btn1_printFullName_Click(object sender, EventArgs e)
    {
        txt_fullName.Text = txt_firstName.Text + " " + txt_lastName.Text;
    }
}
```

Figure 37. Complete code for printing full name.

Please run the application (Ctrl+F5). When you click the Print Full Name button, the full name will be displayed based on what you typed into first name and last name fields. Figure 38 shows an example.

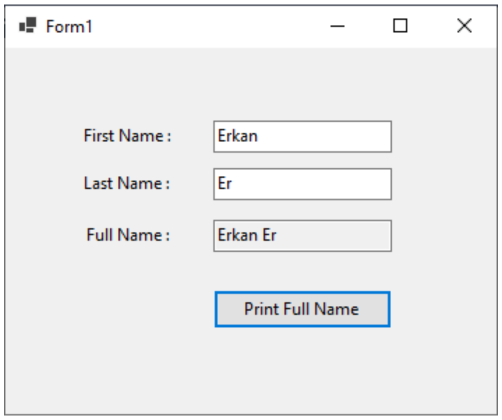


Figure 38. Printing the full name of the user.

This is the end of this module. Please upload *Module3_1* and *Module3_2* project folders as separate zip files to OdtuClass.