

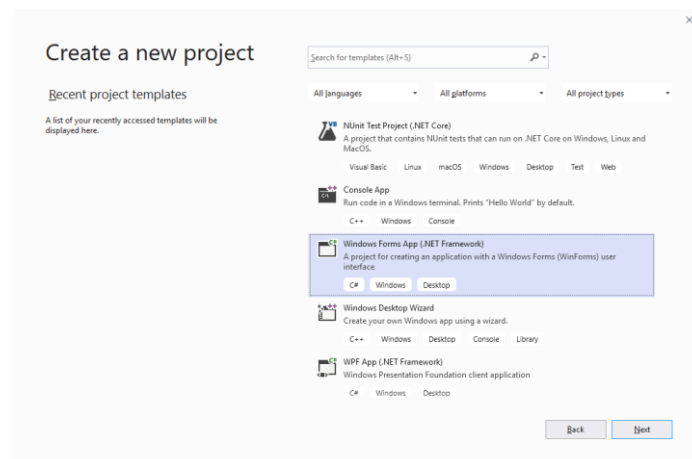
Introduction to the Microsoft Visual Studio IDE

- Become familiar with the IDE
- Become familiar with properties and events of pre-built controls
- Create two applications that calculate an answer from user input

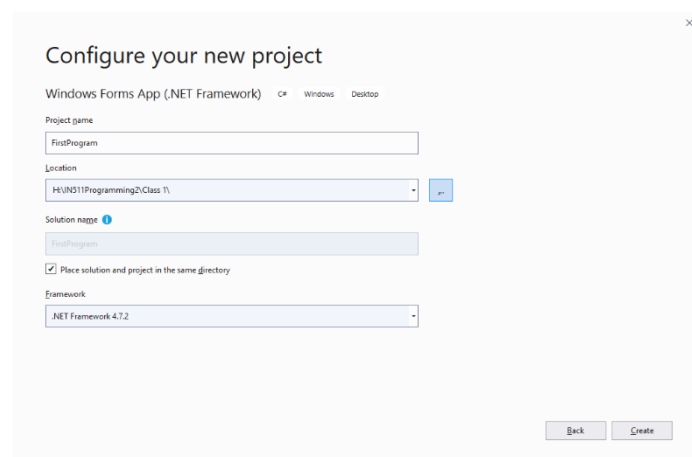
Task 1: Opening a new Form

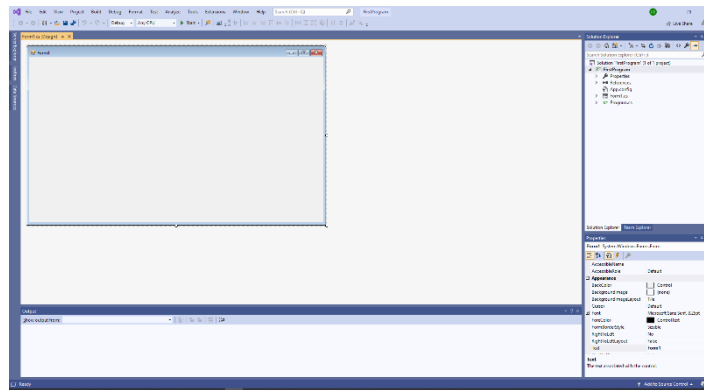
The Microsoft Visual Studio IDE (integrated development environment) is made up of a number of tools that are used to build GUI (Graphical User Interface) applications – those with windows, buttons menus, graphics, etc, by dragging and dropping these pre-built controls onto an existing form as we want it to appear. We build applications that will respond to events – when the user clicks on a button, when a form is closed, etc. Visual Studio fills in all the necessary code for *how* to make these events happen when it compiles our program. We concentrate on writing the code to describe *what* should happen when the user clicks on a button, selects a menu-item, closes a window, and so on. We don't have to worry about all the complex operating system programming underneath – Visual Studio takes care of that for us.

Open *Microsoft Visual Studio*, select *Create a new project*.
Select *Windows Forms App (.NET framework)* then the *Next* button.



Fill in the name and location of your application, and that solution is in same folder as your project, then the *Create* button.





Ensure that these four windows are visible on your screen. Note they are all dockable. The main part is named *Form1.cs(Design)*. This is where, at design time, you create the GUI; it is used as a drawing canvas to resize the main form, add the buttons, labels etc that you will use in your application. This will be the main window that the user sees when your application is run. Your application can contain multiple windows (or Forms) that are displayed in response to user input, data values, etc. To start with, we will build projects that have only one Form.

Task 2: Adding Controls to a Form

Open View/Toolbox/Common Controls.

1. Add two Buttons, a TextBox, PictureBox and a Label to the form.
 2. Run the program, using the green arrow on the toolbar. Click on one of the Buttons, write in the TextBox. What happens?
 3. Change the size, text and name of each of your Buttons.
 4. Change the text, font and visibility of the Label.
 5. Add an image to the PictureBox. How can you resize the image?
 6. Change the colour of the Form.
-

Task 3: Design Time or RunTime?

We said earlier that TextBoxes could be used to show messages to the user of your program. For this to be useful, you need to be able to change the Text property of the Edit Box while the program is running, when you won't be able to get at the Properties list. Changing a property while you are creating your application is called "modifying at design-time". Changing the property from within the program while the program is running is called "modifying at run-time".

Modifying at run-time is very easy: you simply write an assignment statement to assign a new value to the property. To refer to an object's property at run-time you use "dot notation", which you have seen before when referring to elements of records at runtime. For example, to assign the text "Here is my new text" to your TextBox, you put this statement in your code:

```
textBox1.Text = "Here is my new text";
```

Assume you have three controls button1, label1 and textBox1 on your Form. What will be the effect of each of these statements at run-time?

1. button1.Width = 400;
 2. label1.Visible = false;
 3. textBox1.Text = "Go Otago";
 4. button1.Text = label1.Text;
-

Naming Conventions

Use meaningful, descriptive words.

Maximum of 31 characters (they *can* be longer but everything else is ignored).

Use only contain letters, numbers, but *must* begin with a letter.

Do not use underscores.

Cannot contain spaces or any punctuation.

Must *not* be a "reserved word".

Reserved Words

abstract	as	base	bool	break	byte
case	catch	char	checked	class	const
continue	decimal	default	delegate	do	double
else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto
if	implicit	in	int	interface	internal
is	lock	long	namespace	new	null
object	operator	out	override	params	private
protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string
struct	switch	this	throw	true	try
typeof	uint	ulong	unchecked	unsafe	ushort
using	virtual	void	volatile	while	

Class, Method, Namespace and Property names

Use Pascal Casing - first letter of all words are uppercase; all other characters are lower case.

Use a noun or noun phrase to name a class.

Use a verb or action clause to name a method. A method name should tell you what the method does.

eg. Button, PictureBox

Local variables, Objects, Private data items and Parameter names

Use Camel Casing - first letter of all words, except first word, are uppercase; all other characters are lower case.

eg. button1, pictureBox1

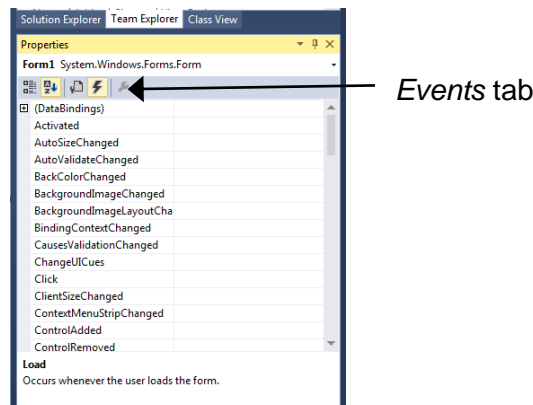
Constants

Use all uppercase letters.

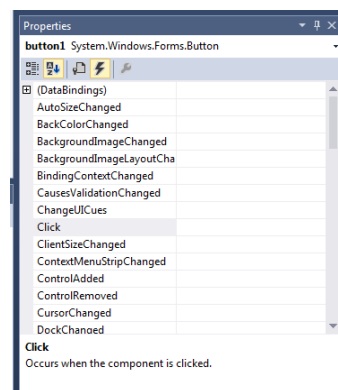
eg. const int LENGTH = 6;

Task 4: Responding to Events

We have seen that controls have Properties. But there is also a tab for Events, the lightning bolt in the Properties window.



If you select a *Button* on your *Form*, and click on the *Events* tab, it will look like this:



This is a list of all the *Events* – things the user can do (or the program can do) – that the Button control understands. For each of these events, you can specify some C# code that should be executed if the event occurs. For example, the Button understands the *Click* event. Whatever code you assign to that event (we'll see how to do this in a moment), will be executed when the user clicks on the Button. The Button also has a *MouseHover* event. This event occurs when the user passes the mouse over the Button. The *DragDrop* event occurs when the user drags the Button on the screen. And so forth.

Until you write code for any event, nothing happens when the event occurs. For example, if you haven't assigned any code to the *Click* event, the user can click on the *Button* as much as he likes, and it has no effect. You saw this happen in Task 2 above.

The code that you assign to an event is called an Event Handler. The code you assign to the *Click* event is called the *button1_Click* handler. Each event handler is simply a C# method. The code for your event handler goes in the *Form1.cs*.

Imagine that you have several Buttons on your Form, and you want different things to happen when you click each Button. A separate *Click* handler is associated with each *Button*, enforced by the naming convention of event handlers.

For example, if my Form is named *Form1*, and my Button is named *button1*, then the code that I want to execute when the user clicks on *button1* should be in a method called

```
private void button1_Click(object sender, EventArgs e)
```

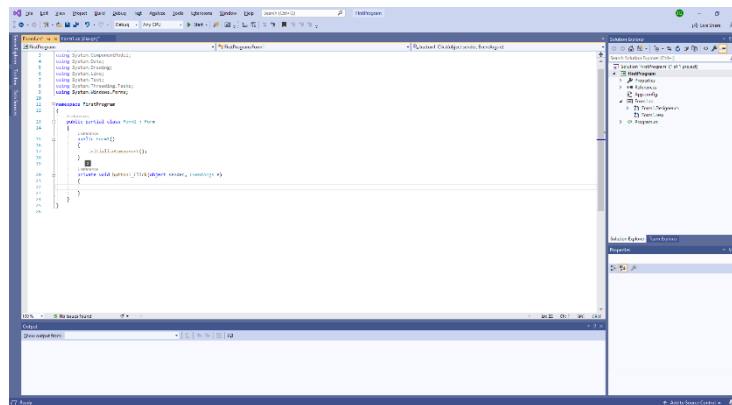
Fortunately, you don't have to worry about typing out these complicated method names, because C# will do it for you. If you double-click in the space beside any event on the *Events* page, three things happen:

- The correct method name is entered into that space
- You are automatically moved from the *Form* into the code.
- C# creates a skeleton for your event handler with the correct method name, and curly brackets, and places the cursor after the first curly bracket, ready for you to start typing your code.

It can be confusing to find yourself suddenly looking at the code instead of the *Form*. Don't be alarmed. This is the same thing that occurs when you toggle between the two views – C# has just done it for you.

Task 5: Responding to Events

1. Start a new project.
2. Place a *Button* and a *Label* on the *Form*.
3. Select the *Button*. Select the *Events* tab. Double-click on the space beside *Click*. You'll see this:



Enter some code for the *Button* to execute. For example, when the *button1* is clicked, the caption property of the *Label* is changed. Here is one possibility:

```
label1.Text = "My text has changed";
```

Run your program and click on the *Button* to see the *Label* change.

4. Add a *Textbox* to your *Form*. Set its text property to 0. Create a variable *nClicks*. The *Click* handler for a *Button* is shown below. What would be the effect of clicking on this *Button* ten times?

```
private void button1_Click(object sender, EventArgs e)
{
    int nClicks = Convert.ToInt16(textBox1.Text);
    nClicks = nClicks + 1;
    textBox1.Text = Convert.ToString(nClicks);
}
```

Note: The `Convert.ToInt16(n)` function translates the parameter value *n* into its 16 bit integer representation. Thus `Convert.ToInt16(5)` converts the string "5" into the integer value 5.

Similarly the `Convert.ToString()` method converts from integer to the equivalent string representation.

Note: The `Convert.ToString` method could also be replaced by the `.ToString()` method.

Note: To increment the variable `nClicks` by one, you can write:

```
nClicks = nClicks + 1;
```

or:

```
nClicks ++;
```

5. Add a *TextBox* to your *Form*. As we saw earlier, the contents of a *TextBox* can be changed while the program is running. Modify your *button1_Click* handler, so that when the *button1* is clicked, the caption of the *label1* changes to whatever is in *textBox1*. Run the program and change the *Label* several times by modifying what is in *textBox1*, then clicking the *button1*.
6. Modify the *button2_Click* handler so that, on each click of the *button2*, *label1* contains a random number between 0 and 99.

Task 6: Sum and Average of Five Numbers

Write a program that allows the user to enter a number into each of five *TextBoxes*. Provide *Buttons* that compute the sum and the average of the five numbers.

Remember that you must use convert the contents of a *TextBox* to a integer, and convert a integer back to a string that can be displayed in a *TextBox*. A possible form layout is:

SumAverage

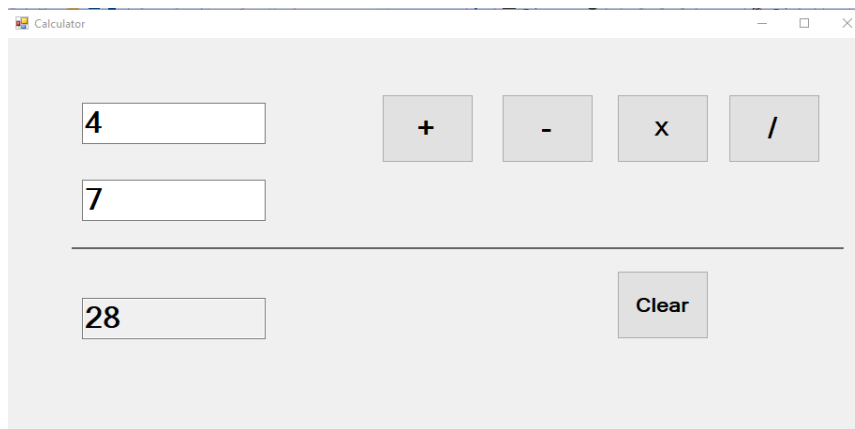
Please enter five numbers:

3	Sum	5267
78	Average	1053
41		
5089		
56		

Task 7: Build your own Calculator

***** CHECKPOINT 1 *****

Write a program that allows the user to enter two numbers, select an arithmetic operation and calculate the answer. Note that this calculator accepts the data as a text entry inside a *TextBox*, rather than as a series of push-button clicks.



1. Start a new application.
2. Set up the *Form*, selecting an appropriate colour, font and style. Set the form's *Text* property to Calculator.
3. Add the *Buttons*, providing the values for each text property.
4. The line between the top two *TextBoxes* and the third *TextBox* is a *Panel*. Decide how you want your interface to look and use the appropriate control.
5. Add the code for the *Click* event for each button.
6. Set the third *TextBox's ReadOnly* property to True. This will stop the user entering a value in this *TextBox* and overwriting the calculated answer.
7. Add MOD and DIV buttons with the appropriate functionality.

Note: To make this application robust, so that it doesn't break under different conditions, we need to check for data validation on the input (checking that the user has entered a valid number.) This is beyond our expertise at present, so we will assume the best-case scenario, that the user enters only valid integers.
