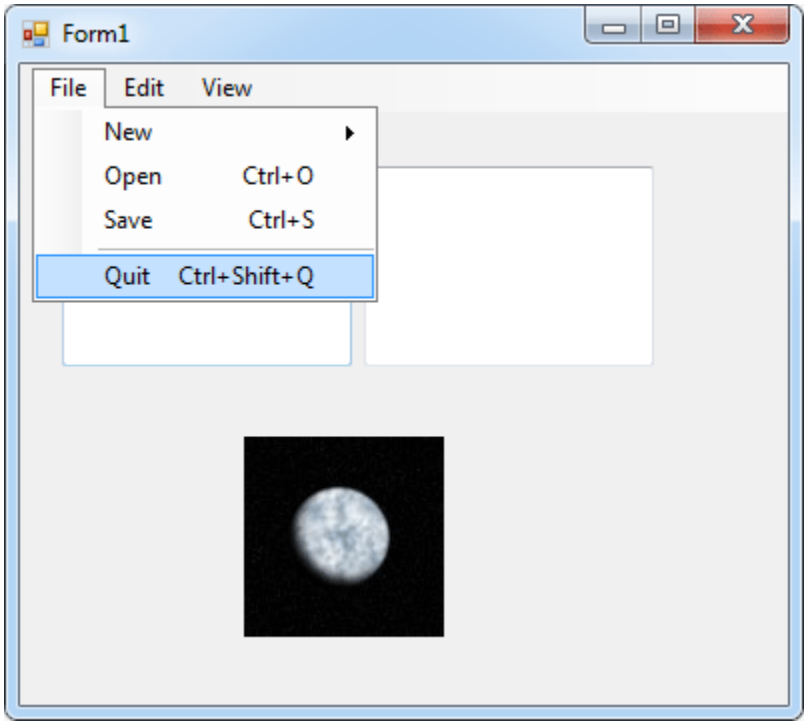
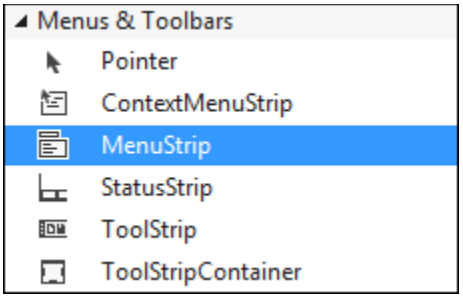
Adding Menus To Windows Forms In C#

In this section, we'll show how to add menus to your forms. You'll add File, Edit, and View menus, with items on each menu, and even sub menus. Here's what you will create:

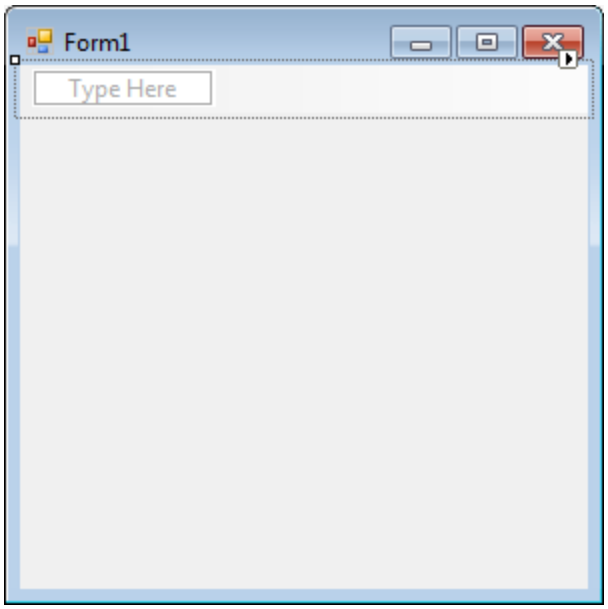


So start a new project by clicking **File > New Project** from the menu at the top of Visual C#. Create a new **Windows Application** project. Call it anything you like. When your new form appears, you can add a menu bar quite easily.

Have a look at the Toolbox on the left of Visual C#. As well as the Common Control tools, there is a section for **Menus and Toolbars**. Click the plus symbol next to this to see the following:



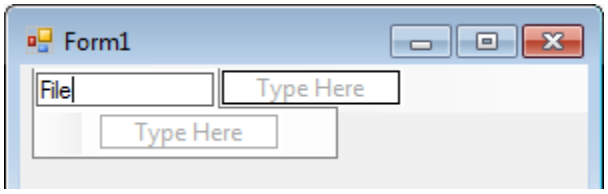
The one you want is **MenuStrip**, which is highlighted in the image above. Double click MenuStrip and you'll see a menu bar appear at the top of your form:



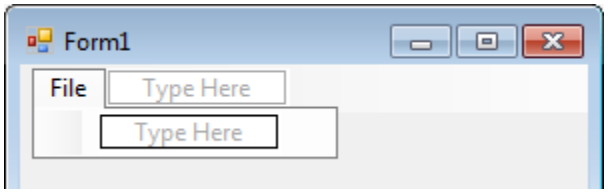
But notice what has appeared at the bottom of your Visual C# window:

This is the MenuStrip object itself. The default Name for the MenuStrip is **menuStrip1**. If your MenuStrip is not selected, you can click on this icon at the bottom. When you do, you'll see all the Properties for the MenuStrip appear in the Properties Window on the right hand side of Visual C#.

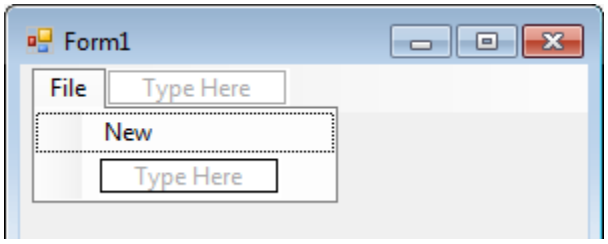
Adding items to your menus is quite simple. Click inside of the area at the top, where it says "Type Here". Now type the word **File**.



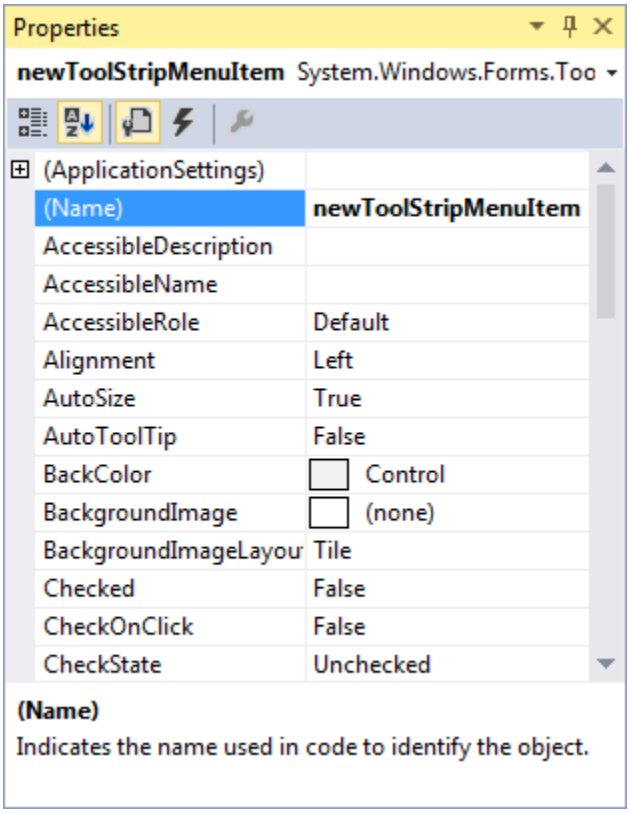
Hit the Enter key on your keyboard and your menu will look like this:



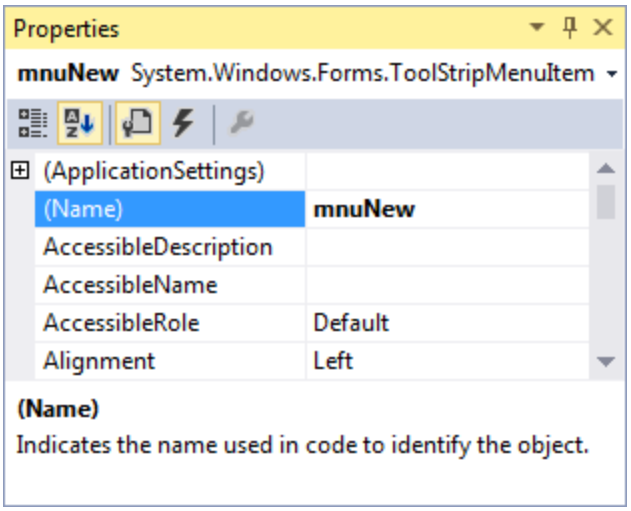
What you have done is to create the main menu item. To add items to your File menu, click inside of the second "Type Here" area pictured above. Now type the word **New**. Hit the Enter key on your keyboard to add the menu item:



Click back on the word New after you have hit the enter key. This will select just this menu item, and no other. Once you create a menu item it has its own Properties that you can change. With the **New** item selected, have a look at the Properties Window on the right hand side of Visual C#:

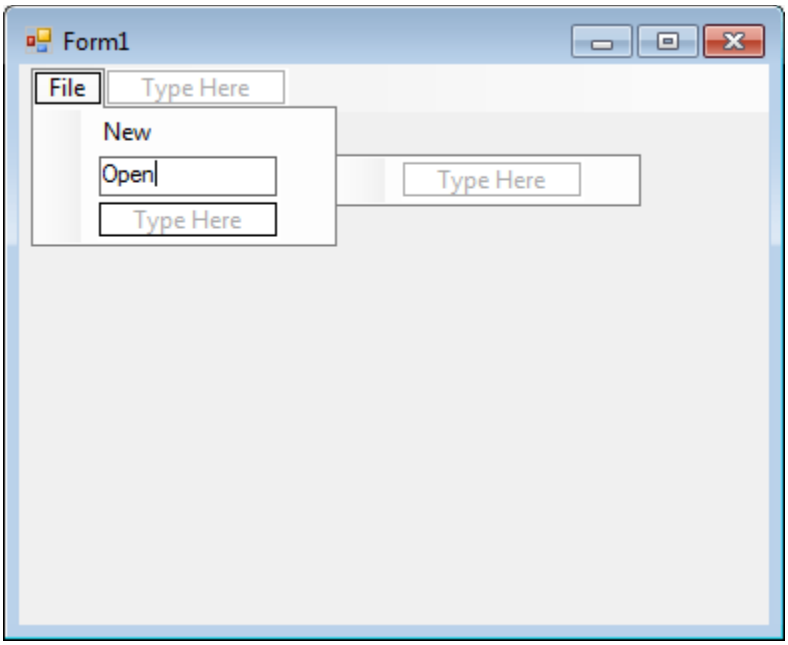


The Property we're interested in is the **Name**. It's a bit too long at the moment. So change it to **mnuNew**, as in the image below:



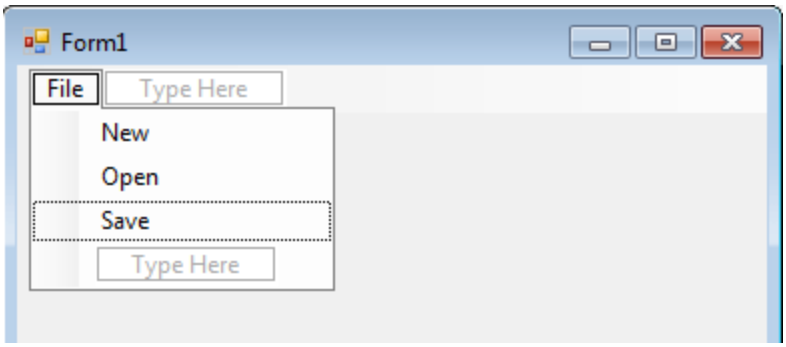
If you scroll down, you'll also see a **Text** property. It will say **New**, at the moment. It says New because that's what you typed in the menu bar when you created this item. You could change this here, if you wanted to. But leave the Text property on New.

Click back on your menu at the top of your form, and then into the "Type here" area just below New. Type the word Open:

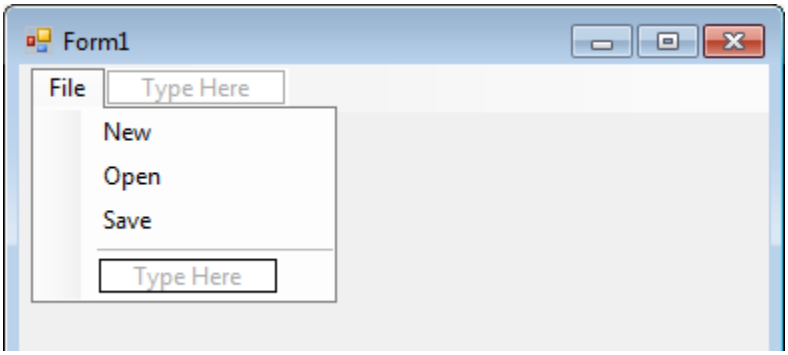


Hit the Enter key on your keyboard to create the Open menu item. Change its **Name** property, just like you did for the New item. Change the name to **mnuOpen**.

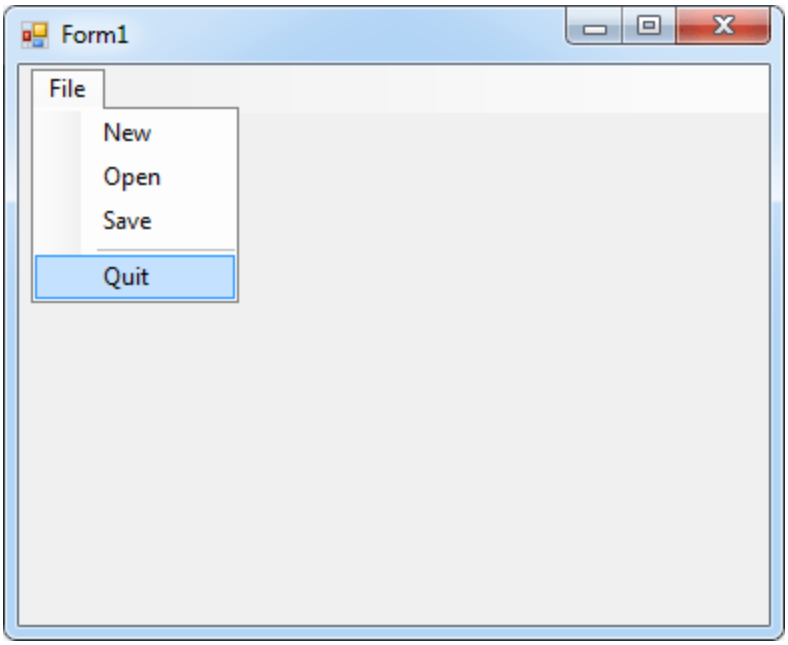
Create a Save menu item, underneath Open. Change its Name property to **mnuSave**. Your File menu will then look like ours below:



We'll now create just two more menu items, a dividing line, and a Quit item. To create a dividing line, click inside of the "Type Here" area below Save. Now type a hyphen (just to the right of the zero key on UK keyboard). When you press the Enter key, C# will turn the hyphen into a dividing line. It should look like this:



Add the **Quit** item below your dividing line. Change the Name property to **mnuQuit**. Your File menu is now complete. To see what it looks like, run your programme. You should have a blue toolbar running across the top with a File menu. Click your File menu:

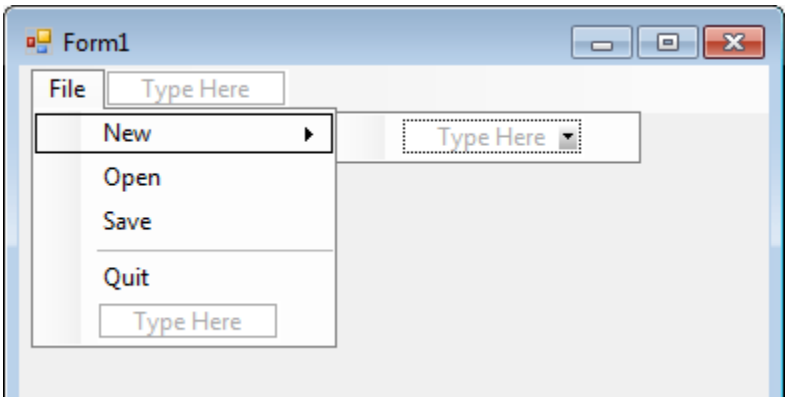


Of course, none of the menus work, because you haven't written any code for them yet. We'll do that soon.

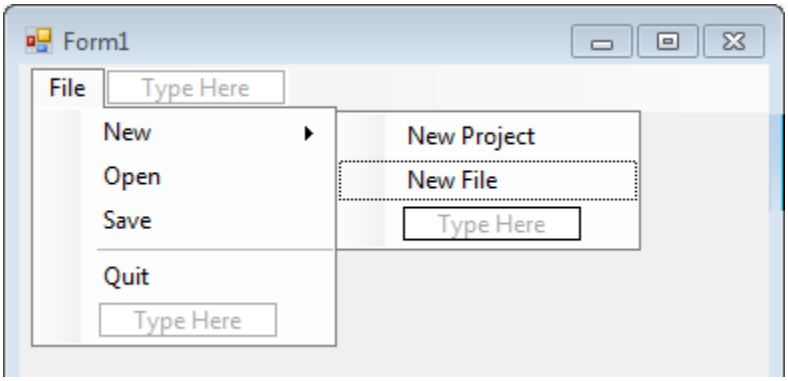
Sub Menus In C# .Net

You can add Sub Menus just as easily. A Sub menu is one that opens out from a main menu item.

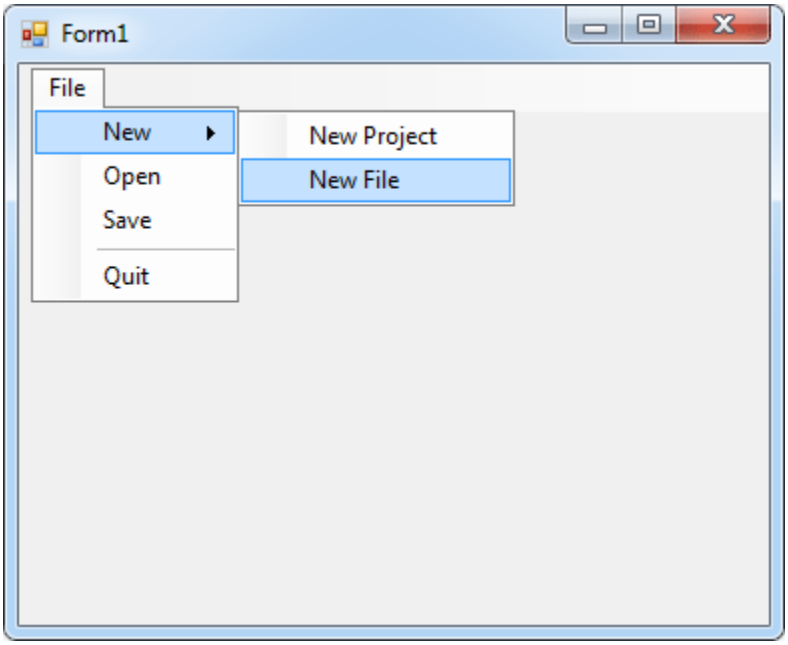
Halt your programme and return to your form. Click on the **New** item to select it. You should see a "Type Here" box appear to the right of New:



Click Inside of this box and type **View Project**. Hit the enter key and type **View Files** in the box below this. Your menu will then look like this:



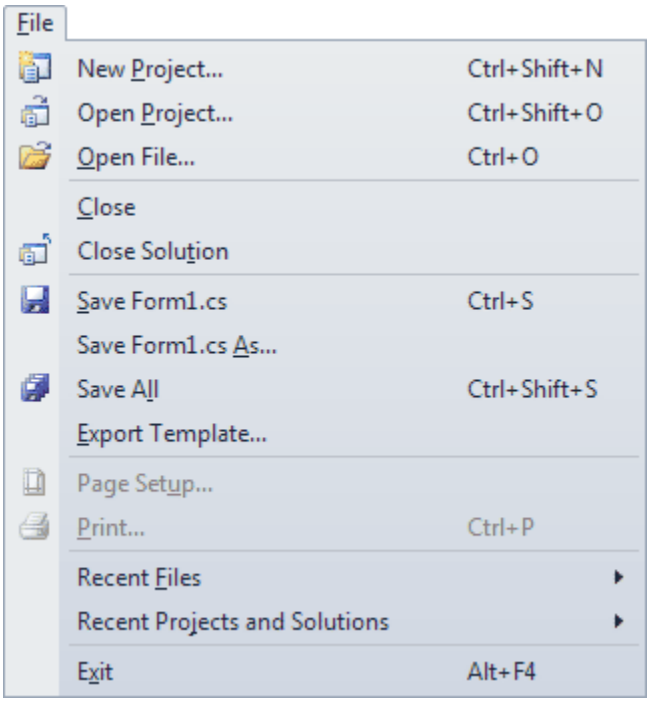
When the form is run, the Sub Menu will look like this:



Sub menus are quite easy to add! In the next lesson, you'll learn how to add shortcuts to your menu items.

Menu Shortcuts In C# .Net

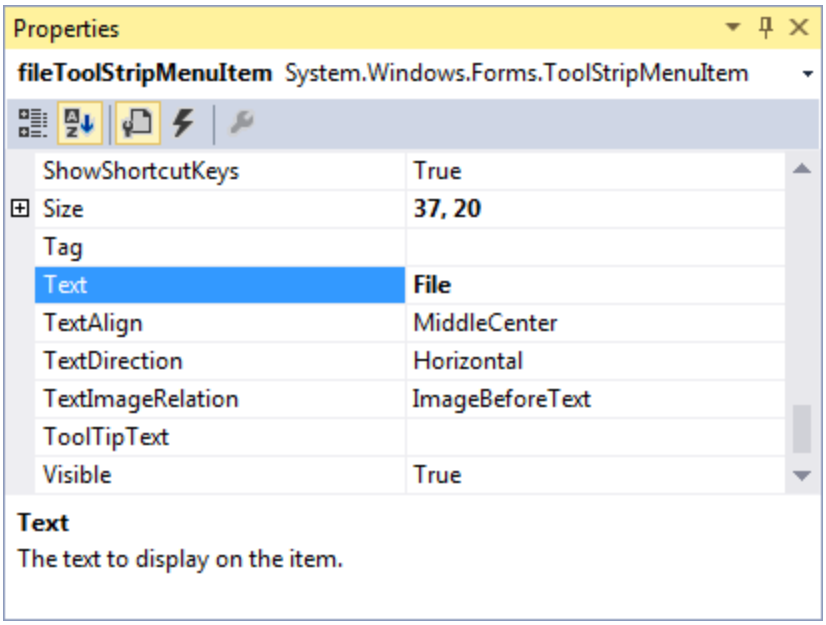
Menus usually have shortcuts. These are the underlined letters that you see when you click a menu. They sometimes have a shortcut key combination to the right of the menu item. For example, here's the File menu from Visual C# with all the underlines and key combinations showing:



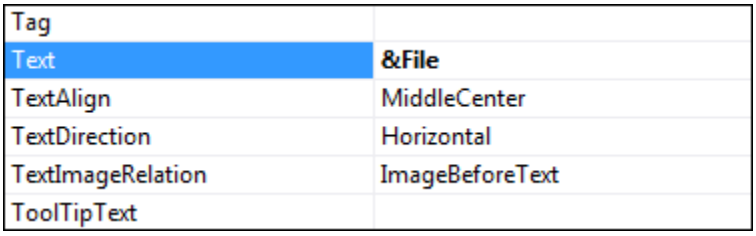
To see these, you need to hit the ALT key on your keyboard. When you see the underlined letters, press the key that corresponds to the underlined letter. Pressing the "F" key, for example, will then cause the menu to drop down. Pressing any of the underlined letters on the File menu will implement that menu item. (In our edition of Visual C# Express, pressing the letter P only switches back and forward between the first two items. The other letters work OK, though.)

You can also use the key combinations to the right of the menu item. Holding down CTRL + SHIFT + N at the same time will cause the New Project dialogue box to appear.

To add shortcuts to your own menus, click your **File** item to select it. Now have a look at its Properties in the Property Window. Scroll down until you locate the text item:

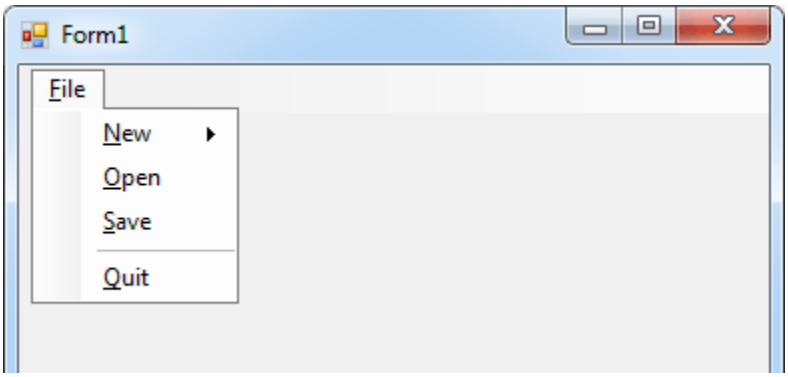


To add an underline to any of the letters, you use the ampersand symbol (&) before the letter you want to use as a shortcut. In the image below, we've added an ampersand just before the "F" of File:



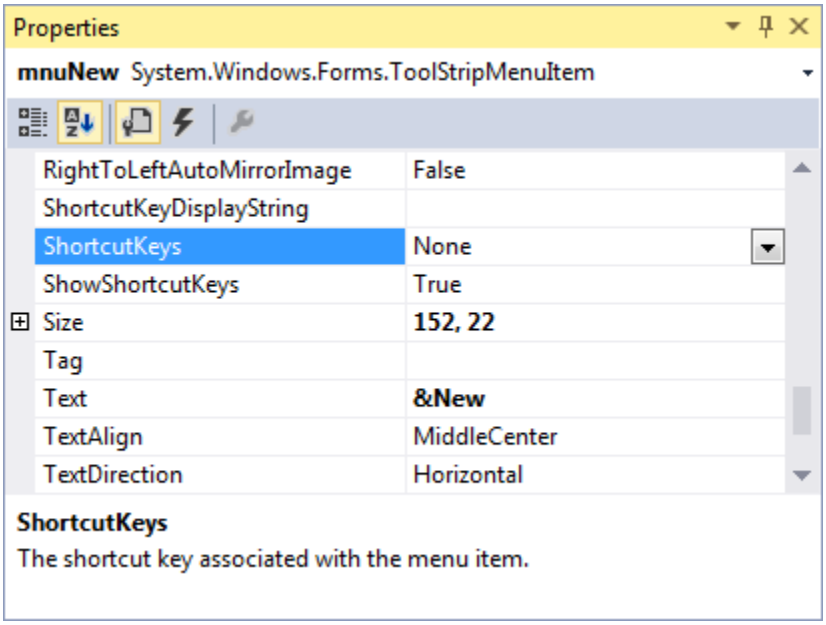
And here's what the menu looks like with the ampersand added:

As you can see, there's now a line underneath the letter "F". In the next image, we've added more underlines to the rest of the File menu:

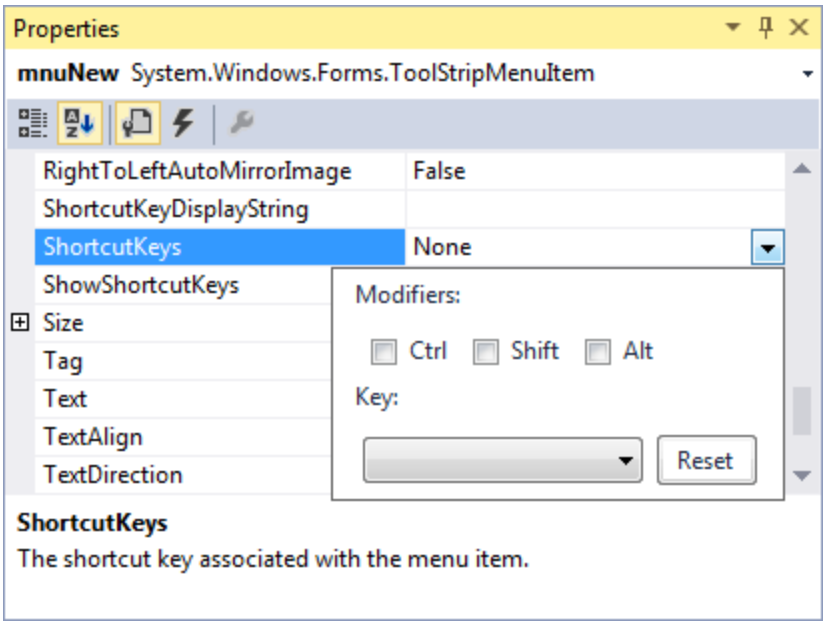


Add the same underlines to your own File menu. Remember: click a menu item to select it, locate the text property, and add an ampersand before the letter you want to use as a shortcut. When you run the programme, don't forget to press the ALT key on your keyboard, otherwise you won't see the underlined letters.

The key combination shortcuts are just as easy to add. Click on your **New** menu item to select it. Locate the **ShortcutKeys** Property in the Properties Window:

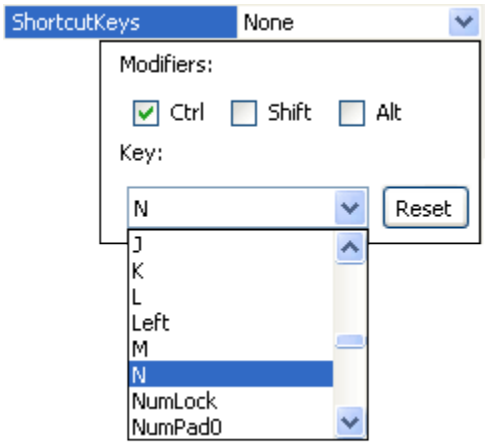


At the moment, it's set to None. Click the down arrow to see the following options:

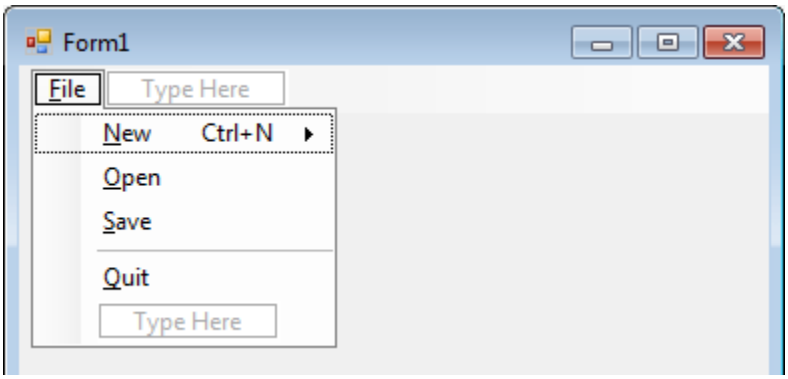


The Modifiers are the CTRL, Shift, and ALT keys. You can select one or all of these, if you want. To activate a shortcut, you would then have to hold down these keys first. So if you want your users to hold down the CTRL and Shift keys, plus a letter or symbol, then you would check the relevant Modifier boxes above.

The letters and symbols can be found on the Key drop down list. Click the down arrow to see the following:

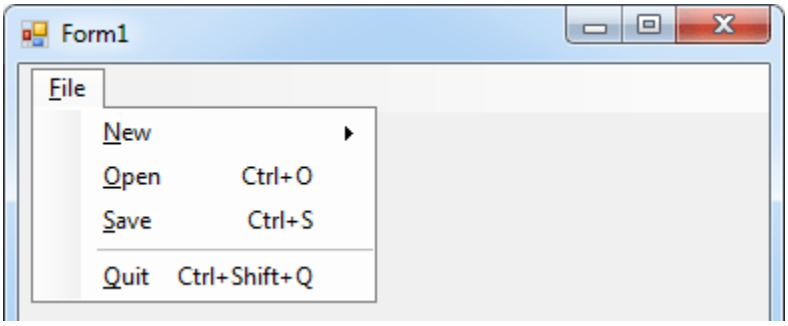


In the image above, we've gone for the CTRL modifier, and the letter "N". Clicking back on the menu, here's what it now looks like:



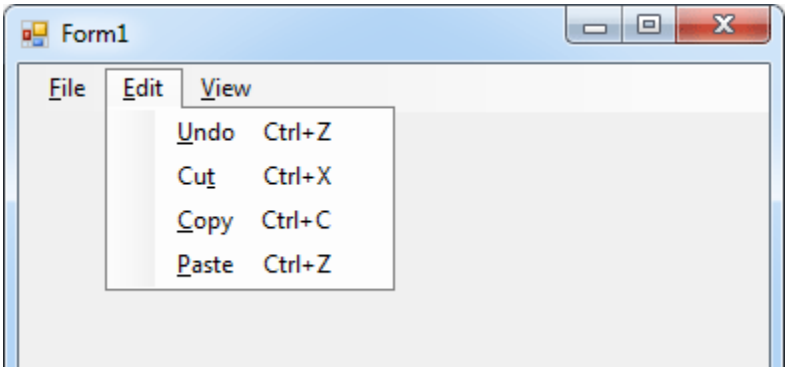
As you can see, the shortcuts for the New menu item are an Underline, and Ctrl + N.

Have a look at the next image, and add the same Shortcut Keys to your File menu:



The ones you are adding are the final three: Open, Save and Quit. We'll get to coding the menu items shortly, but here's an exercise to complete. (Don't skip this exercise because you'll need the menu items!)

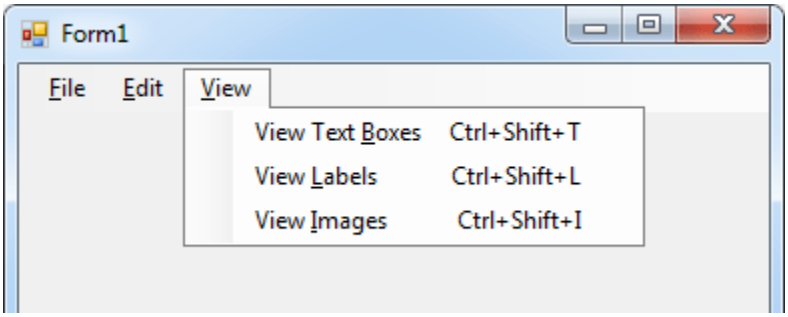
Exercise
Add an Edit menu to your menu bar with the following items:



Include the underline shortcuts, and the key combination shortcuts. For the Name Property of each menu item, use the following:

Undo: mnuUndo
Cut: mnuCut
Copy: mnuCopy
Paste: mnuPaste

Exercise
Add a View menu to your menu bar with the following items:



Again, include the underline and key combination shortcuts. Set the Name Property of your menu items to the following:

View Text Boxes: mnuViewTextBoxes
View Labels: mnuViewLabels
View Images: mnuViewImages

OK, it's now time to do some coding for the menu items you have created.

C# Code For Your Quit Menu

Of course, a menu is no good if nothing happens when you click an item. So we need to add code behind the menu items. We'll start with the Quit item, which should be on your File menu. There's only one line of code for this.

Return to your form, and click the menu strip. Click the **File** item to see its menu. Double click on your **Quit** item and the coding window should open. Your cursor should be flashing between the curly brackets of the Quit code stub:

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

Notice that the Name you gave your menu item is used in the code stub: **mnuQuit**. But when a user clicks your Quit menu, you want the programme to end. To close down a Windows application, you can use this:

Application.Exit( );

So add that line of code between the curly brackets of your Quit code stub. Run your programme and test it out. Hit the CTRL and SHIFT keys, and then the letter Q on your keyboard. The programme should close straight away. It does this because of the key combination shortcuts you added.

To see your underline shortcuts in action, start your programme again. Press the ALT key on your keyboard and you should see all the underlines appear for File, Edit and View. Press the letter "F" on your keyboard (the underlined letter), and the menu should drop down. Now press the letter Q on your keyboard. Because this is the underlined letter, the programme should exit.

You can add more code to menu items - anything you like, in fact. Something you do see on Quit menus is a message box:

"Are you sure you want to Quit?"

To add a message box to your own code, try this:

```
if (MessageBox.Show("Really Quit?", "Exit", MessageBoxButtons.OKCancel) ==
DialogResult.OK)
{

    Application.Exit();

}
```

The first two lines should be one line in your code. It's only on two lines here because there's not enough room for it on this page. But in between the round brackets of an if statement, we have a message box:

MessageBox.Show("Really Quit?", "Exit", MessageBoxButtons.OKCancel)

This will get you a dialogue box with OK and Cancel buttons. C# will wait until the user clicks a button. Our code uses an if statement to test which button was clicked. To see which one the user clicked, you add this on the end:

== DialogResult.OK

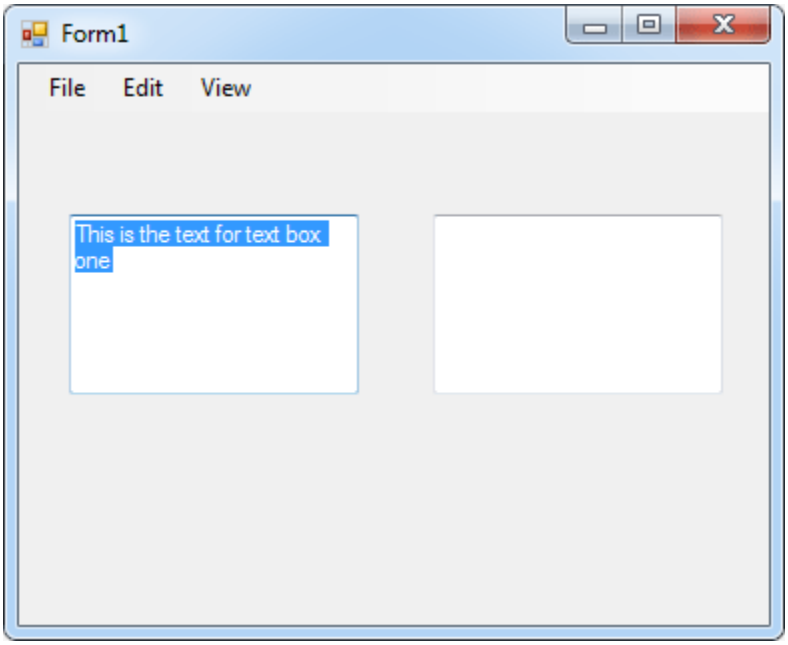
So the line reads "IF the result of the dialogue box was that the OK button was clicked, then Exit the Application."

In the next part, you'll see how to code for your Edit menu.

The Edit Menu

We'll leave the rest of the File menu till last, as it's a bit more complicated. Our Edit menu is not too difficult, as there's only one line of code for each menu item.

To see Cut, Copy, Paste and Undo in action, add two text boxes to your form. Set the **MultiLine** Property of each text box to true. This will allow you to have a text box with more than one line of text. For text box one, type anything you like for the Text property. Your form should then look like this:



What's we'll do now is to enable the Cut, Copy and Paste menu items, as well as the undo. We'll first Cut the highlighted text from the text box and then Undo the operation.

So return to your Form. Click your blue menu bar at the top, and click on your Edit menu. Double click the **Cut** item. C# will create a code stub for you. If you gave your Cut menu item the Name of **mnuCut** then your code stub will look like this:

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

The code to cut any highlighted text is quite simple. Add this between the curly brackets of your mnuCut code:

textBox1.Cut( );

Cut( ) is a method that's built in to C#. It works on text boxes, amongst other things, and does what it says - cuts!

Before you try it out, return to your form and double click your **Undo** menu item. Add the following line for the code:

```
textBox1.Undo( );
```

Now try it out. Run your form and highlight the text in the text box. Use your Cut menu to cut the text. Then use your Undo menu to restore the text.

You can also check to see if any text was selected. Change your code to this:

```
if (textBox1.SelectedText != "")
{

    textBox1.Cut();

}
```

We're using an if statement to check a property of text boxes called **SelectedText**. This can tell you if any text is selected. We're using the "Does Not Equal" operators (!=) followed by a pair of double quotes. A pair of double quotes with no spaces means that it's a blank string of text. If no text was selected, then the if statement is **true**. In which case the Cut operation can go ahead.

You can manipulate selected text with the SelectedText Property. In the code below, we're handing the selected text to a string variable and displaying it in a message box:

```
string someText;

if (textBox1.SelectedText != "")
{

    someText = textBox1.SelectedText;
    MessageBox.Show(someText);

}
```

For the Undo menu, you might want to check if the operation can actually be Undone. If you want to check for this, there is another property of text boxes called **CanUndo**. You use it like this:

```
if (textBox1.CanUndo == true)
{

    textBox1.Undo();

}
```

Only if the operation can be Undone will the code for the if statement execute.

However, if you run your programme and cut some text, clicking Undo twice will first restore the text and then cut it again! (You're undoing the restore.) To remedy this, you can clear the undo operation. Change your code to this. The new line is in black bold text below:

```
if (textBox1.CanUndo == true)
{

    textBox1.Undo();
    textBox1.ClearUndo();

}
```

So you just add the ClearUndo() method after the dot of textBox1. Try it again and you'll find that clicking Undo twice won't cut the text again.

Copy And Paste In C# .Net

To Copy something to the Clipboard, you highlight text and click a Copy item on an Edit menu. Once the data is copied to the Clipboard, it can be Pasted elsewhere. We'll implement this with our menu system.

Double click the **Copy** item on your **Edit** menu. You'll be taken to the code stub for your Copy menu item. Add the following code between the curly brackets:

```
textBox1.Copy();
```

Using the Copy method of text boxes is enough to copy the data on to the Windows Clipboard. But you can first check to see if there is any highlighted text to copy. Change your code to this:

```
if (textBox1.SelectionLength > 0)
{

    textBox1.Copy();

}
```

We're using an if statement again. This time, we are checking the **SelectionLength** property of text boxes. The length returns how many characters are in the text that was selected. We want to make sure that it's greater than zero.

We'll use the second text box to Paste. So access the code stub for your Paste menu item, using the same technique as before. Add the following between the curly brackets of your Paste code:

```
textBox2.Paste();
```

Notice that we're now using textBox2 and not textBox1. After the dot, you only need to add the Paste method.

Try your Edit menu out again. Highlight the text in the first text box. Click **Edit > Copy**. Now click into your second text box and click **Edit > Paste**.

You can also check to see if there is any data on the Clipboard, and that it is text and not, say, an image. Add this rather long if statement to your code:

```
if (Clipboard.GetDataObject().GetDataPresent(DataFormats.Text) == true)
{

    textBox2.Paste();
    Clipboard.Clear();

}
```

Getting at the data on the Clipboard can be tricky, but we're checking to see what the DataFormat is. If it's text then the if statement is true, and the code gets executed. Notice the last line, though:

```
Clipboard.Clear();
```

As you'd expect, this Clears whatever is on the Clipboard. You don't need this line, however, so you can delete it if you prefer. See what it does both with and without the line.

The View Menu

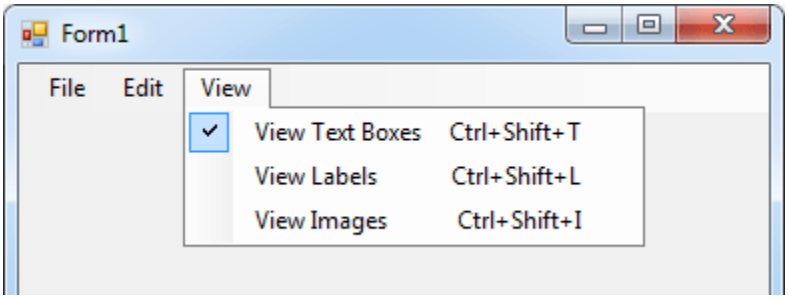
We have three items on our **View** menu. But we'll only implement two of them. For the first one, View Text Boxes, we'll show you a handy programming technique with Boolean variables - how to toggle them on and off.

So return to your form, and double click the menu item for **View Text Boxes**. C# will generate the code stub for you:

```
private void mnuViewTextboxes_Click(object sender, EventArgs e)
{

}
```

What we'll do is to hide the text boxes when the menu item is clicked, and unhide the text boxes when you click again. A check icon will then appear or disappear next to the menu item. Here's an image of what we'll be doing:



To place a check mark next to a menu item, you use the **Checked** Property of the menu item. Add this to your View Textboxes code stub, in between the curly brackets:

mnuViewTextboxes.Checked = true;

So you just type a dot after the Name of your menu item. Then select the **Checked** property from the IntelliSense list. Checked is a Boolean value that you either set to true or false (it's either got a check mark next to it or it hasn't).

Run your programme and click your **View Textboxes** menu item. You should see a check appear. It does so because the default value for the Checked property is false. It only becomes true when you click the menu item, thereby running the code you added.

The question is, how do you get the Check mark symbol to disappear when it's clicked again? Obviously you need to set it to false, meaning not checked. But what's the code?

A handy programming technique is to toggle Boolean values off and on. You do it with the aid of the NOT operator (!). Amend your code to this:

mnuViewTextboxes.Checked = !mnuViewTextboxes.Checked;

So instead of setting the Checked value to true, we have this:

!mnuViewTextboxes.Checked;

This says, "NOT Checked". But doesn't mean "Unchecked". What you are doing is setting the Boolean variable to what it is currently NOT. Remember: Checked can either be true OR false.

So if Checked is currently true, set it to false, and vice versa. The result then gets stored back in the Property on the left of the equals sign.

Run your programme and try it out. Click the menu item to see the Check symbol. Click it again and it will disappear. This toggling of Boolean variables is quite common in programming, and can save you a lot of tricky coding!

To actually do something with the text boxes, though, you can add an if statement to examine whether the variable is true. What we'll do is make the text boxes visible if there's a Check, and not visible if there isn't a Check. Add this code just below the line you already have:

```
if (mnuViewTextboxes.Checked)
{

    textBox1.Visible = true;
    textBox2.Visible = true;

}
else
{

    textBox1.Visible = false;
    textBox2.Visible = false;

}
```

The Property we are changing is the **Visible** Property of text boxes. As its name suggests, this hides or un-hides an object. Again, it's a Boolean value, though. So we could have just done this:

```
textBox1.Visible = !textBox1.Visible;
```

The use of the NOT operator will then toggle the Visibility on or off. We added an if statement because it's handy to actually examine what is in the variable, rather than just assuming.

One line you may puzzle over is this:

```
if (mnuViewTextboxes.Checked)
```

The part in round brackets could have been written like this, instead:

```
if (mnuViewTextboxes.Checked == true)
```

For if statements, C# is trying to work out if the code in round brackets is true. So you can leave off the "== true" part, because it's not needed. If you want to check for false values, you can use the NOT operator again. Like this:

if (!mnuViewTextboxes.Checked)

This is the same as saying this:

if (mnuViewTextboxes.Checked == **false**)

Using the NOT operator is considered more professional. They mean the same, though, so use which one is better for you.

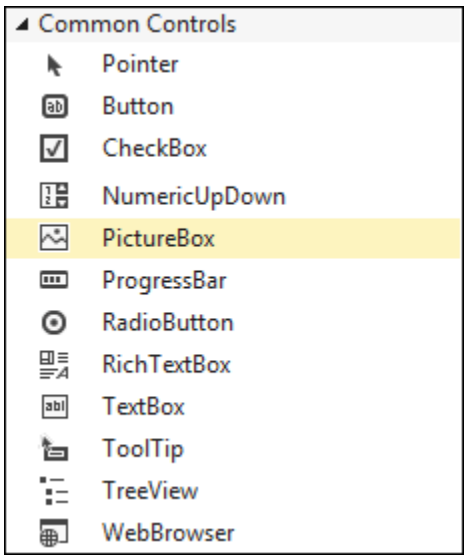
Adding Images In C# .Net

You have seen the Open File dialogue box countless times. It's the one that appears whenever you click **File > Open** on a Windows machine. You then navigate through folders, searching for the file you want to open. For our View Images menu, we'll do something slightly more complex - we'll have our own Open File dialogue box that allows you to select images from your computer. When you select an image, it will then appear in a new control on your form.

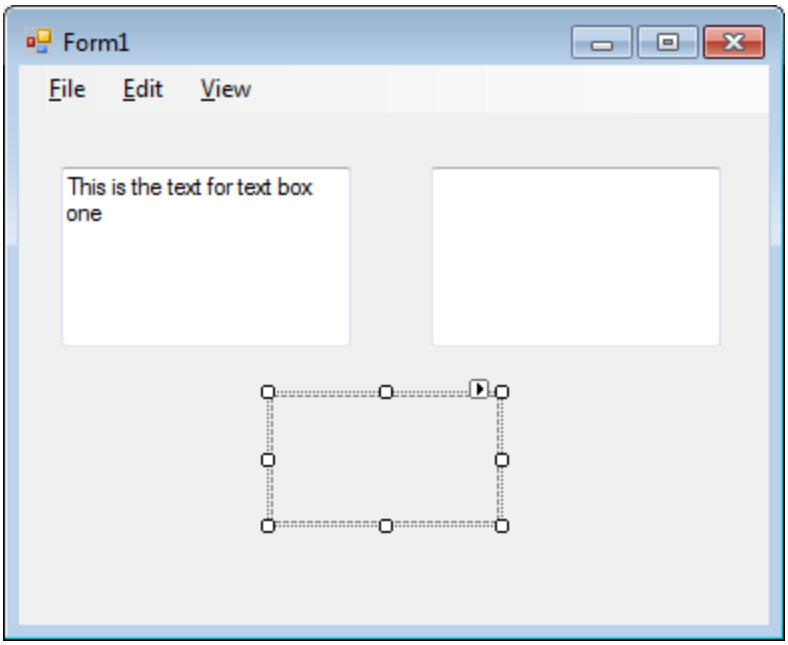
So we need a place on our form where we can store images. We'll use a Picture Box.

Have a look at the Toolbox on the left hand side of Visual C#. Under **Common Controls**, locate **PictureBox**:

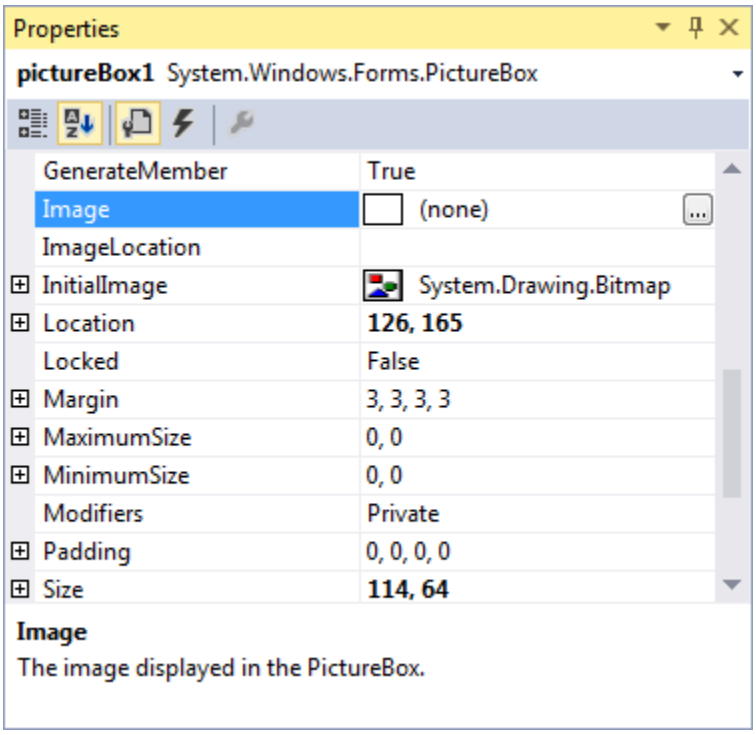
Once you've select the PictureBox tool. Click on your form once to add a new PictureBox control. Your form should then look like this:



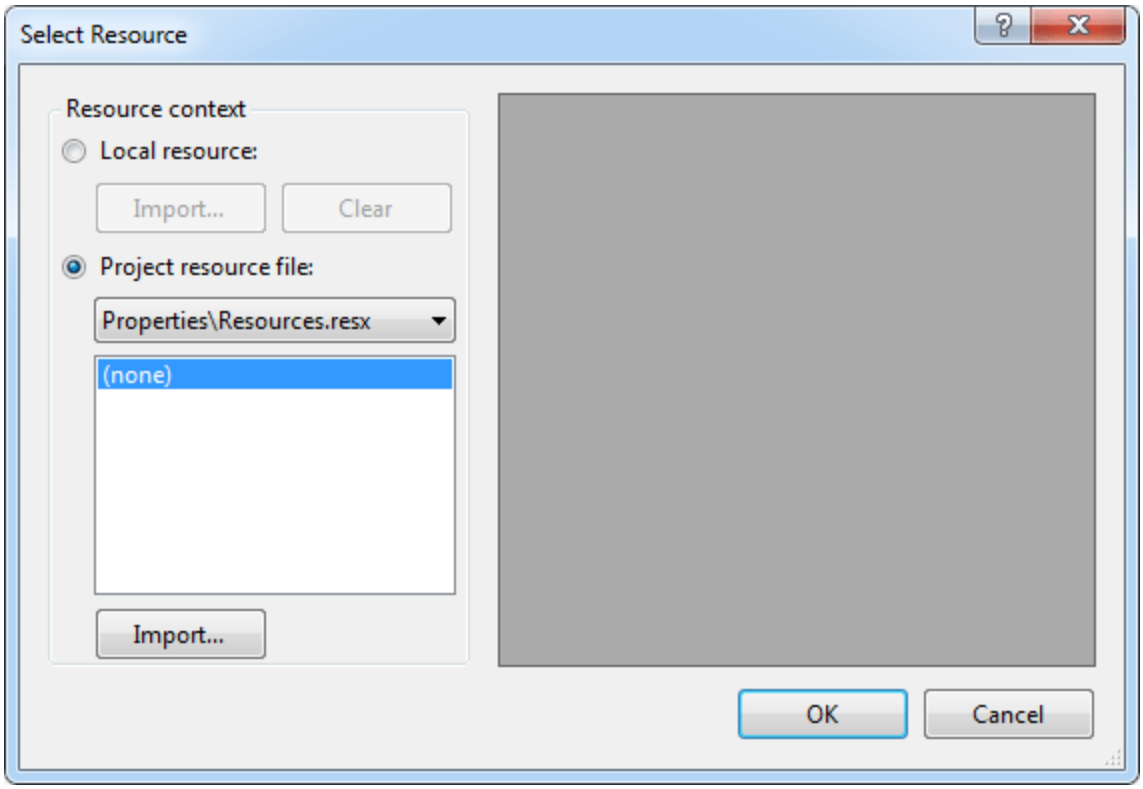
Once you've select the PictureBox tool. Click on your form once to add a new PictureBox control. Your form should then look like this:



The PictureBox control is blank when you first add one. To add an image to it at Design Time, have a look at the Properties Window. Locate the **Image** property:

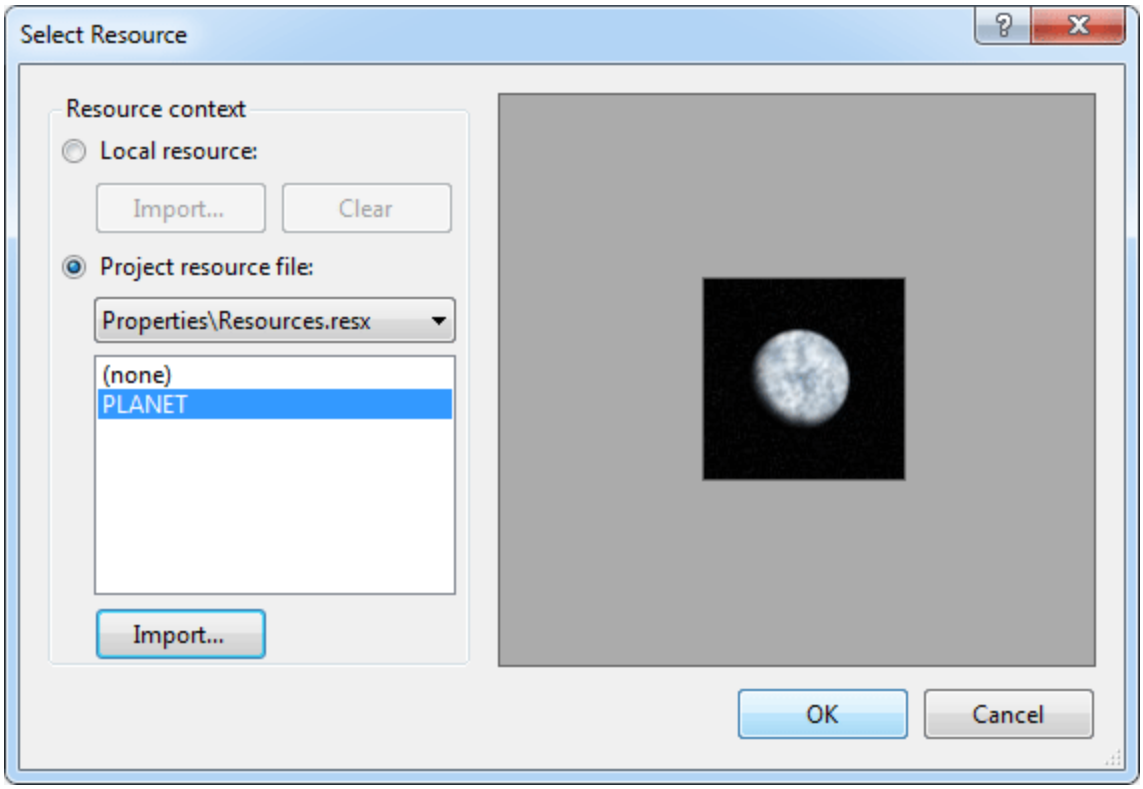


Click the button with the three dots on it to see a dialogue box appear:

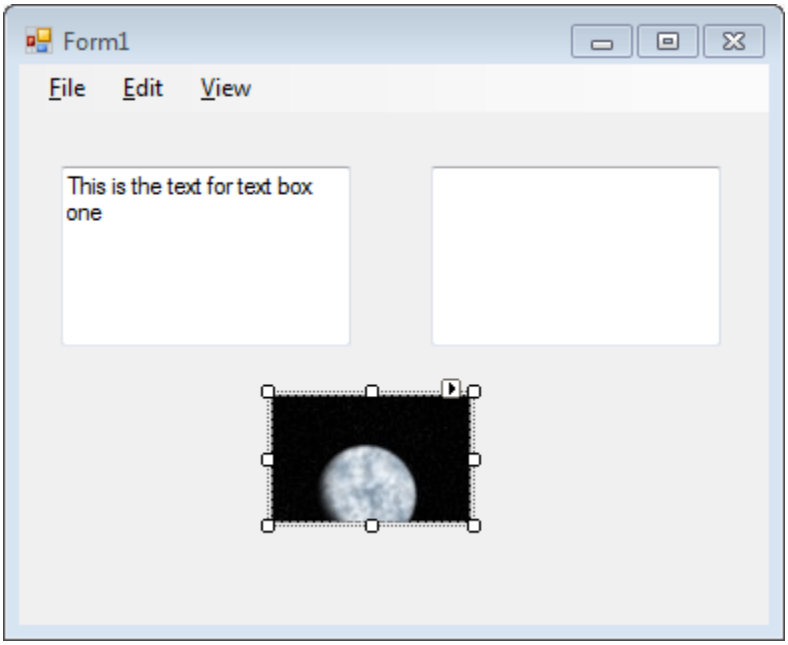


Click the Import button at the bottom and you'll see a standard Open dialogue box. Search your hard drive for a suitable image. Because you have "Project resource file" selected, C# will copy the image to a folder in your project. (This is handy if you want to send your programme to anyone else.)

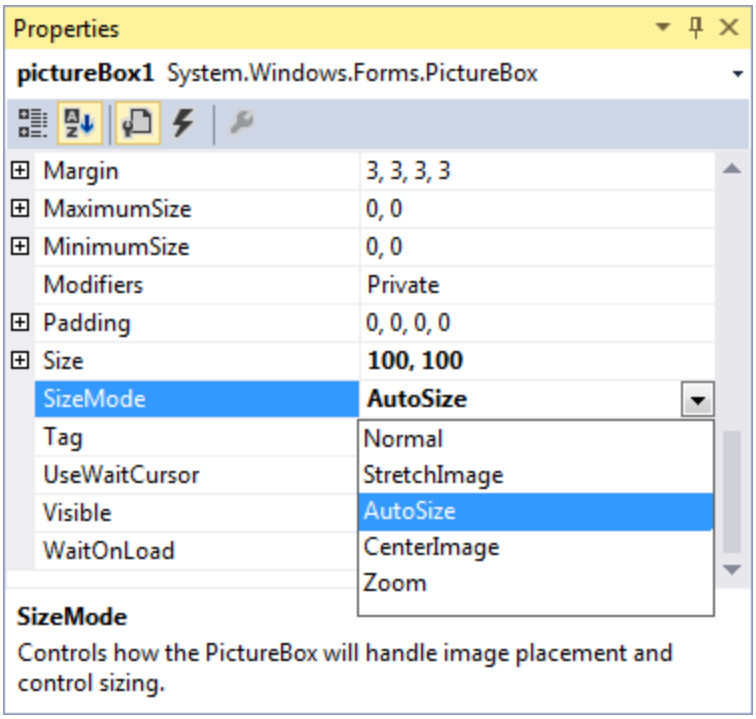
In the image below, we've gone for a picture of a planet:



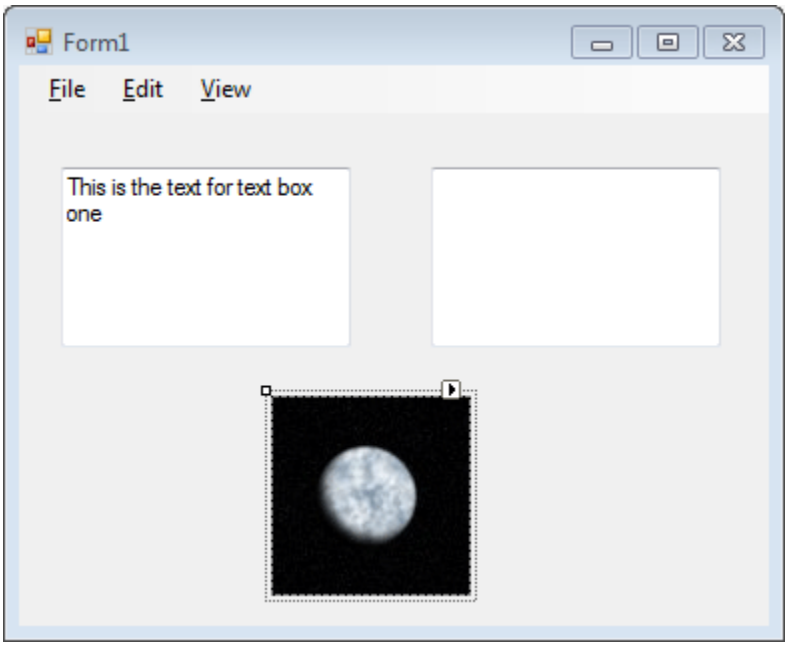
Click OK and you'll be taken back to your form:



Notice that the image is too big for the PictureBox. Locate the **SizeMode** property in the Properties Window:



As you can see, there are a few to choose form. Select **AutoSize**, and the PictureBox will automatically stretch to the size of your image:



If you run your programme, you'll see the image appear on the form. It won't have a border, though. If you want a border, explore the **BorderStyle** Property of your PictureBox control.