To create a Windows Forms application project

1. On the **File** menu, click **New Project**.
2. If you’re not using Visual Studio Express, you need to select a language first. From the **Installed Templates**list, select either **C#** or **Visual Basic**.
3. Click the **Windows Forms Application** icon, enter **PictureViewer** as the name, and then click **OK**. Visual Studio automatically creates the solution for you.
4. On the **File** menu, click **Save All**, or on the toolbar, click the **Save All** button, which appears as follows.

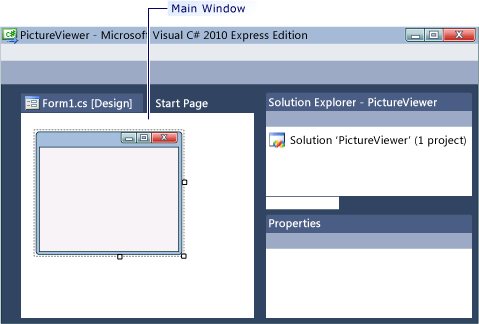
Save All toolbar button

Save All toolbar button

|  |
| --- |
| **NoteNote** |
| Visual Studio saves the project in your projects folder. The integrated development environment (IDE) automatically fills in the folder name and project name. If you’re using Visual Studio Express, you need to complete steps 5-7. For non-Express versions of Visual Studio, a project is saved when it's first created, so steps 5-7 aren't necessary. |

1. Make sure the project is saved in a folder under your **My Documents** folder.
2. Verify that the **Create Directory for Solution** check box is selected.
3. Click **Save**.

|  |
| --- |
| **NoteNote** |
| When you create your project, the Visual Studio IDE automatically creates several files for you and puts them in a folder. You can explore those files using the **Solution Explorer** window. When you first create your project, the files are saved in a temporary folder. Clicking the **Save All** button tells the IDE to copy them to a permanent folder (which is typically under your **My Documents** folder). |

1. You may have noticed that the words *solution* and *project* have different meanings, which will be explained later in this tutorial.
2. The following picture shows what your IDE window should look like.
3. IDE window
4. 
5. If your screen doesn't look like the preceding picture, on the **Window** menu, click **Reset Window Layout**. If any of the windows are missing, on the **View** menu, click **Properties Window** or **Solution Explorer**. If there's an extra window open, click the **Close** (x) button in the upper right.
6. Look at the picture. From the upper left and going clockwise, the picture shows:
   * **Main window**   This is where you do most of your work. You use this window to work with forms and edit code. Currently, it shows a form in the Form Editor. At the top of the window, there are two tabs: the **Start Page** tab and the **Form1.cs [Design]** tab. (In Visual Basic, it's .vb instead of .cs.)
   * **Solution Explorer window**   This is where all the files in your solution appear. If you click a file, the information inside the **Properties** window changes. If you double-click a code file (which ends in .cs in Visual C# and .vb in Visual Basic), the code file or a designer for the code file opens.
   * **Properties window**   This is where you change the properties of items that you select in the other windows.

To run your program

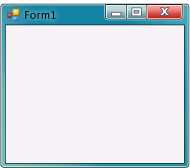
1. Press the F5 key or click the **Start Debugging** toolbar button, which appears as follows.

Start Debugging toolbar button

Start Debugging toolbar button

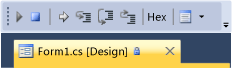
1. The IDE runs your program, and a window appears. The following picture shows the program you just built. The program is running, and you will soon add to it.

Windows Form application program running



1. Go back to the IDE, and look at the new toolbar.

Debugging toolbar



1. Click the square **Stop Debugging** button or from the **Debug** menu, click **Stop Debugging**. The program stops running, and the window closes. You can also just close the open window to stop debugging.

|  |
| --- |
| **NoteNote** |

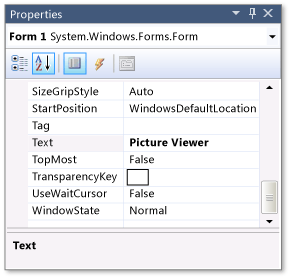
To set your form properties

1. Be sure you're looking at Windows Forms Designer. In the IDE, click the **Form1.cs [Design]** tab (or the**Form1.vb [Design]** tab in Visual Basic).
2. Click anywhere inside the form to select it. Look at the **Properties** window, which should now be showing the properties for the form. Forms have various properties. For example, you can set the foreground and background color, title text that appears at the top of the form, size of the form, and other properties.

|  |
| --- |
| **NoteNote** |
| If the **Properties** window doesn't appear, stop your program by clicking the square **Stop Debugging**button (or just close the window). |

1. After the form is selected, scroll down to the bottom of the **Properties** window, and find the **Text** property. Click **Text**, type **Picture Viewer**, and then press ENTER. Your form should now have the text **Picture Viewer** in its title bar, and the **Properties** window should look like the following picture.

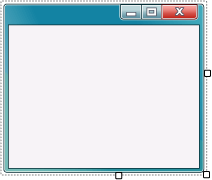
Properties window



|  |
| --- |
| **NoteNote** |
| Properties can be ordered by a Categorized or Alphabetical view. You can switch between these two views by using the buttons on the **Properties** window. In this tutorial, it's easier to find properties through the Alphabetical view. |

1. Go back to Windows Forms Designer. Click the form's lower-right drag handle, which is the small white square in the lower-right of the form and appears as follows.

Drag handle



Drag it to resize the form so the form is wider and a bit taller.

1. Look at the **Properties** window, and notice that the **Size** property has changed. The **Size** property changes each time you resize the form. Try dragging the form to resize it to a form size of approximately 550, 350, which should work well for this project.
2. Run your program again. Press the F5 key or click the **Start Debugging** toolbar button, which appears as follows.

Start Debugging toolbar button

Start Debugging toolbar button

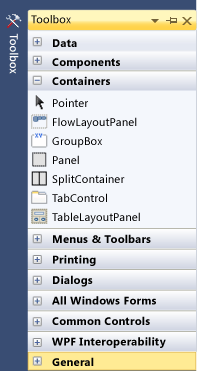
Just like before, the IDE builds and runs your program, and a window appears.

1. Before going to the next step, stop your program, because the IDE won't let you change your program while it's running.

### To lay out your form with a TableLayoutPanel control

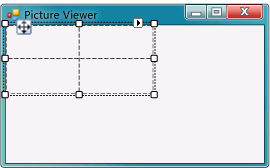
1. Go to Windows Forms Designer. Look at the left side of the form and locate the **Toolbox** tab. Point to and pause on the **Toolbox** tab, and the Toolbox appears. (Or, from the **View** menu, click **Toolbox**.)
2. Click the plus sign next to the **Containers** group to open it, as shown in the following picture.

Containers group



1. You can add controls like buttons, check boxes, and labels to your form. Double-click the **TableLayoutPanel**control in the Toolbox. When you do this, the IDE adds a **TableLayoutPanel** control to your form, as shown in the following picture.

TableLayoutPanel control



|  |
| --- |
| **NoteNote** |
| After you add your TableLayoutPanel, if a window appears inside your form with the title**TableLayoutPanel Tasks**, click anywhere inside the form to close it. You will learn more about this window later in the tutorial. |
| **NoteNote** |
| Notice how the Toolbox expands to cover your form when you click its tab, and closes after you click outside of it. That's the IDE auto-hide feature. You can turn it on or off for any of the windows by clicking the pushpin icon in the upper-right corner of the window to toggle auto-hide and lock it in place. The pushpin icon appears as follows. |

Pushpin icon

Pushpin icon

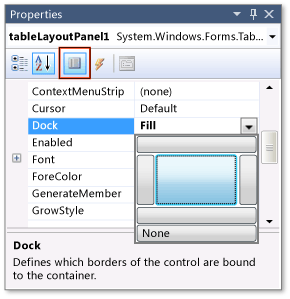
1. Be sure **TableLayoutPanel** is selected by clicking it. You can verify what control is selected by looking at the drop-down list at the top of the **Properties** window, as shown in the following picture.

Properties window showing TableLayoutPanel control



1. The control selector is a drop-down list at the top of the **Properties** window. In this example, it shows that a control called **tableLayoutPanel1** is selected. You can select controls either by clicking in Windows Forms Designer or by choosing from the control selector. Now that **TableLayoutPanel** is selected, find the **Dock**property and click **Dock**, which should be set to **None**. Notice that a drop-down arrow appears next to the value. Click the arrow, and then select the **Fill** button (the large button in the middle), as shown in the following picture.

Properties window with Fill selected



1. After you set the TableLayoutPanel **Dock** property to **Fill**, the panel fills the entire form. If you resize the form again, the TableLayoutPanel stays docked, and resizes itself to fit.

|  |
| --- |
| **NoteNote** |
| A TableLayoutPanel works like a table in Microsoft Office Word: It has rows and columns, and an individual cell can span multiple rows and columns. Each cell can hold one control (like a button, a check box, or a label). Your TableLayoutPanel will have a **PictureBox** control spanning its entire top row, a **CheckBox**control in its lower-left cell, and four **Button** controls in its lower-right cell. |
| **NoteNote** |
| Although it was stated that each cell can hold only one control, the lower-right cell has four **Button**controls. This is because you can put a control in a cell that holds other controls. That kind of control is called a container, and the TableLayoutPanel is a container. You will learn more about this later in the tutorial. |

1. Currently, the TableLayoutPanel has two equal-size rows and two equal-size columns. You need to resize them so the top row and right column are both much bigger. In Windows Forms Designer, select the TableLayoutPanel. In the upper-right corner, there is a small black triangle button, which appears as follows.

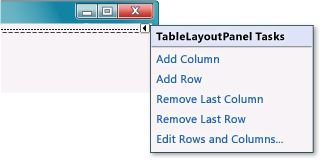
Triangle button

Triangle button

This button indicates that the control has tasks that help you set its properties automatically.

1. Click the triangle to display the control's task list, as shown in the following picture.

TableLayoutPanel tasks



1. Click the **Edit Rows and Columns** task to display the **Column and Row Styles** window. Click **Column1**, and set its size to 15 percent by being sure the **Percent** button is selected and entering **15** in the **Percent** box. (That's a **NumericUpDown** control, which you will use in a later tutorial.) Click **Column2** and set it to 85 percent. Don't click the **OK** button yet, because the window will close. (But if you do, you can reopen it using the task list.)
2. From the **Show** drop-down list at the top of the window, click **Rows**. Set **Row1** to 90 percent and **Row2** to 10 percent.
3. Click **OK**. Your TableLayoutPanel should now have a large top row, a small bottom row, a small left column, and a large right column. You can resize the rows and columns in the TableLayoutPanel by dragging their borders.

### To continue or review

* To go to the next tutorial step, see [Step 5: Add Controls to Your Form](http://msdn.microsoft.com/en-us/library/dd492173(v=vs.100).aspx).
* To return to the previous tutorial step, see [Step 3: Set Your Form Properties](http://msdn.microsoft.com/en-us/library/dd492146(v=vs.100).aspx).

## Community Additions

[ADD](http://msdn.microsoft.com/en-us/library/community/add/dd492143(v=vs.100).aspx)

### Changing row and column values.

When changing the size (percentage-wise) of the rows and columns, do NOT change the numerical values manually by writing into the text box! Use the up-down buttons at the side.

[http://i1.social.s-msft.com/profile/u/avatar.jpg?displayname=auliver&size=small](http://social.msdn.microsoft.com/profile/auliver/)

[Auliver](http://social.msdn.microsoft.com/profile/auliver/)

12/12/2013

### No contradiction

RobCM - if you read the note below the note you quote you will see:

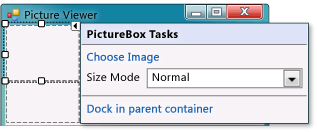
"Although it was stated that each cell can hold only one control, the lower-right cell has four   
Button  
 controls. This is because you can put a control in a cell that holds other controls. That kind of control is called a container, and the TableLayoutPanel is a container. You will learn more about this later in the tutorial."

So you can place more than one control in a cell by using a container as stated.

To add controls to your form

1. Go to the Toolbox and expand the **Common Controls** group. This shows the most common controls that you see on forms.
2. Double-click the **PictureBox** control. The IDE adds a **PictureBox** control to your form. Because the TableLayoutPanel is docked to fill your form, the IDE adds the **PictureBox** control to the first empty cell.
3. Click the black triangle on the new **PictureBox** control to display its task list, as shown in the following picture.

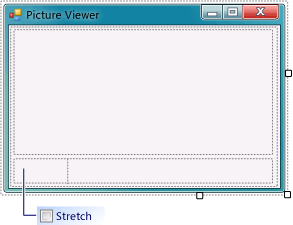
PictureBox tasks



|  |
| --- |
| **NoteNote** |
| If you accidentally add the wrong type of control to your TableLayoutPanel, you can delete it. Right-click the control, and then click **Delete** from the menu. You can also select **Undo** from the **Edit** menu to remove controls from the form. |

1. Click the **Dock in parent container** link. This automatically sets the PictureBox **Dock** property to **Fill**. To see this, click the **PictureBox** control to select it, go to the **Properties** window, and be sure that the **Dock** property is set to **Fill**.
2. Make the PictureBox span both columns by changing its **ColumnSpan** property. Select the **PictureBox** control and set its **ColumnSpan** property to **2**. Also, when the PictureBox is empty, you want to show an empty frame. Set its **BorderStyle** property to **Fixed3D**.
3. Add the **CheckBox** control to the form. Double-click the **CheckBox** item in the Toolbox to make the IDE add a new **CheckBox** control to the next free cell in your table. Because a PictureBox takes up the first two cells, a**CheckBox** control is added to the lower-left cell. Select the new **CheckBox** control and set its **Text** property to**Stretch**, as shown in the following picture.

CheckBox control with Text property set to ‘Stretch’



1. Go to the **Containers** group in the Toolbox (where you got your **TableLayoutPanel** control) and double-click the **FlowLayoutPanel** item to add a new control to the last cell in the PictureBox. Then dock it in the parent container (either by choosing **Dock in parent container** from the task list, or by setting its **Dock** property to**Fill**).

|  |
| --- |
| **NoteNote** |
| A FlowLayoutPanel is a container that arranges other controls in neat rows in order. When you resize a FlowLayoutPanel, if it has room to lay out all of its controls in a single row, it does that. Otherwise, it arranges them in lines, one on top of the other. You will use a FlowLayoutPanel to hold four buttons. |

To add buttons

1. Select the FlowLayoutPanel that you added. Go to **Common Controls** in the Toolbox and double-click the**Button** icon to add a button called **button1** to your FlowLayoutPanel. Repeat to add another button. The IDE determines that there's already a button called **button1** and calls the next one **button2**.

|  |
| --- |
| **NoteNote** |
| In Visual Basic the buttons are named with an initial cap, so **button1** is **Button1**, **button2** is **Button2**, and so on. |

1. Typically, you add the other buttons using the Toolbox. This time, click **button2**, and then on the **Edit** menu, click **Copy** (or press Ctrl+C). On the **Edit** menu, click **Paste** (or press Ctrl+V) to paste a copy of your button. Now paste it again. The IDE has now added **button3** and **button4**.

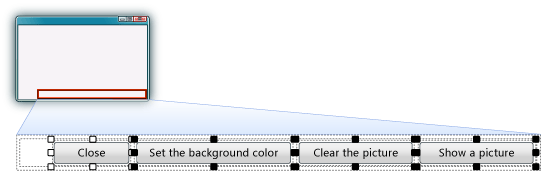
|  |
| --- |
| **NoteNote** |
| You can copy and paste any control. The IDE names and places the new controls in a logical manner. If you paste a control into a container, the IDE chooses the next logical space for placement. |

1. Select the first button and set its **Text** property to **Show a picture**. Then set the **Text** properties of the next three buttons to **Clear the picture**, **Set the background color**, and **Close**.
2. The next step is to size the buttons and arrange them so they align to the left side of the panel. Select the FlowLayoutPanel and look at its **FlowDirection** property. Change it so it's set to **RightToLeft**. As soon as you do, the buttons should align themselves to the right side of the cell, and reverse their order so that the **Show a picture** button is on the right.

|  |
| --- |
| **NoteNote** |
| If the buttons are still in the wrong order, you can drag the buttons around the FlowLayoutPanel to rearrange them in any order. You can click one of the buttons and drag it left or right. |

1. Click the **Close** button to select it. Hold down the CTRL key and click the other three buttons, so that they are all selected. While all the buttons are selected, go to the **Properties** window and scroll up to the **AutoSize**property. This property tells the button to automatically resize itself to fit all of its text. Set it to **true**. Your buttons should now be sized properly and be in the right order. (As long as all four buttons are selected, you can change all four **AutoSize** properties at the same time.) The following picture shows the four buttons.

Picture Viewer with four buttons

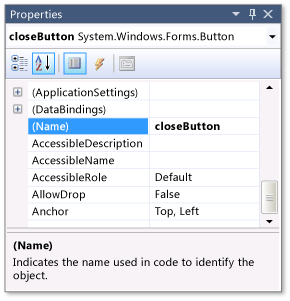


1. Now run your program again to see your newly laid out form. Clicking the buttons and the check box doesn't do anything yet, but it will work soon.

To name your button controls

1. Click the **Close** button. (If you still have all the buttons selected, press the ESC key to cancel the selection.) Scroll in the **Properties** window until you see the **(Name)** property. (The **(Name)** property is near the top when the properties are alphabetical.) Change the name to **closeButton**, as shown in the following picture.

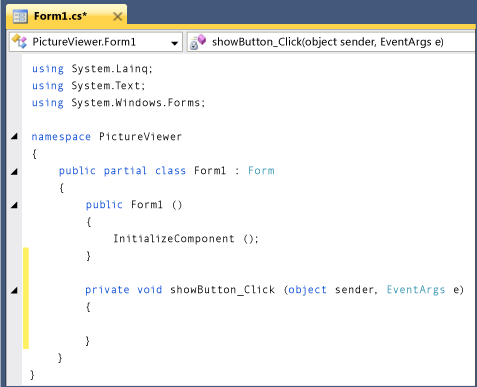
Properties window with closeButton name



|  |
| --- |
| **NoteNote** |
| If you try changing the name of your button to **close** **Button**, with a space between the words close and Button, the IDE displays an error message: "Property value is not valid." Spaces (and a few other characters) are not allowed in control names. |

1. Rename the other three buttons to **backgroundButton**, **clearButton**, and **showButton**. You can verify the names by clicking the control selector drop-down list in the **Properties** window. The new button names appear.
2. Double-click the **Show a picture** button in Windows Forms Designer. When you do, the IDE opens a new tab in the main window called the **Form1.cs** tab, as shown in the following picture.

Form1.cs tab with Visual C# code



1. Focus on this part of the code.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492140(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

private void showButton\_Click(object sender, EventArgs e)

{

}

You are looking at a method called **showButton\_Click()**. The IDE added this when you double-clicked the**showButton** button. This method contains code that runs when the **Show a picture** button is clicked.

|  |
| --- |
| **NoteNote** |
| In this tutorial, the Visual Basic code that's automatically generated has been simplified by removing everything between the parentheses, (). Whenever this occurs, you can remove the same code. Your program will work either way. For the remainder of the tutorials, any automatically generated code is simplified whenever possible. |

1. Go to the designer tab (the **Form1.cs [Design]** tab in Visual C# or the **Form1.vb [Design]** tab in Visual Basic) and double-click the **Clear the picture** button. Repeat this for the last two buttons. Each time, the IDE adds a new method to the form's code.
2. To add one more method, double-click the **CheckBox** control in Windows Forms Designer to make the IDE add a **checkBox1\_CheckedChanged()** method. That method gets called whenever the user selects or clears the check box.

|  |
| --- |
| **NoteNote** |
| When working on a program, you often move between the code editor and Windows Forms Designer. The IDE makes it easy to navigate in your project. Use **Solution Explorer** to open Windows Forms Designer by double-clicking **Form1.cs** in Visual C# or **Form1.vb** in Visual Basic. |

1. The following shows the new code that you see in the code editor.
2. C#
3. [**VB**](http://msdn.microsoft.com/en-us/library/dd492140(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)
4. private void clearButton\_Click(object sender, EventArgs e)
5. {
6. }
7. private void backgroundButton\_Click(object sender, EventArgs e)
8. {
9. }
10. private void closeButton\_Click(object sender, EventArgs e)
11. {
12. }
13. private void checkBox1\_CheckedChanged(object sender, EventArgs e)
14. {
15. }

|  |
| --- |
| **NoteNote** |
| The five methods that you added are called *event handlers*, because your program calls them whenever an event (like a user clicking a button or selecting a box) happens.  When you double-click a control in the IDE, it adds an event handler method for the control. For example, when you double-click a button, the IDE adds an event handler for its Click event (which is called whenever the user clicks the button). When you double-click a check box, the IDE adds an event handler for its CheckedChanged event (which is called whenever the user selects or clears the box).  After you add an event handler for a control, you can return to it at any time from Windows Forms Designer by double-clicking the control. |
| **NoteNote** |
| Names are important when you build programs, and methods (including event handlers) can have any name that you want. When you add an event handler with the IDE, it chooses a name based on the control's name and the event being handled. For example, the Click event for a button named **showButton**is called the **showButton\_Click()** event handler method. Also, opening and closing parentheses () are usually added after the method name to make it clear that methods are being discussed. |

To add dialog components to your form

1. Go to Windows Forms Designer and open the **Dialogs** group in the Toolbox.

|  |
| --- |
| **NoteNote** |
| The **Dialogs** group in the Toolbox has components that open many useful dialog boxes for you, which can be used for opening and saving files, browsing folders, and choosing fonts and colors. You use two dialog components in this project: **OpenFileDialog** and **ColorDialog**. |

1. To add a component called **openFileDialog1** to your form, double-click **OpenFileDialog**. To add a component called **colorDialog1** to your form, double-click **ColorDialog** in the Toolbox. (You use that one in the next tutorial step.) You should see a gray box at the bottom of Windows Forms Designer that has an icon for each of the two dialog components that you added, as shown in the following picture.

Dialog components

Dialog components

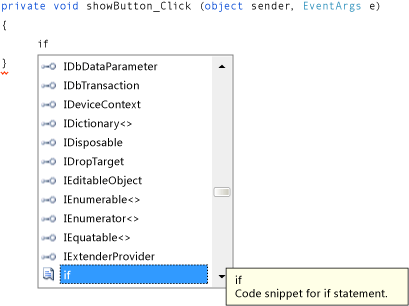
1. Go to Windows Forms Designer and click the **openFileDialog1** icon in the gray box at the bottom of the designer. Set two properties:
   * Set the **Filter** property to the following (you can copy and paste it): JPEG Files (\*.jpg)|\*.jpg|PNG Files (\*.png)|\*.png|BMP Files (\*.bmp)|\*.bmp|All files (\*.\*)|\*.\*
   * Set the **Title** property to the following: Select a picture file

|  |
| --- |
| **NoteNote** |
| To see an example of the **Open File** dialog box in a different application, open Notepad or Paint, and on the **File** menu, click **Open**. Notice how there's a **Files of type** drop-down list at the bottom. You just used the **Filter** property in the **OpenFileDialog** component to set that up. Also, notice how the **Title** and **Filter**properties are bold in the **Properties** window. The IDE does that to show you any properties that have been changed from their default values. |

To write code for the Show a picture button event handler

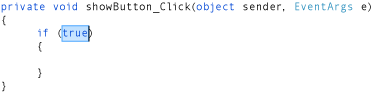
1. Go to Windows Forms Designer and double-click the **Show a picture** button. The IDE immediately goes to the code designer and moves your cursor so it's inside the **showButton\_Click()** method that you added previously.
2. Type an **i** on the empty line between the two braces { }. (In Visual Basic, type on the empty line between Private Sub… and End Sub.) An **IntelliSense** window opens, as shown in the following picture.

IntelliSense with Visual C# code



1. The **IntelliSense** window should be highlighting the word **if**. (If not, enter a lowercase **f**, and it will.) Notice how the yellow tooltip next to the **IntelliSense** window shows **Code snippet for if statement**. (In Visual Basic, the tooltip also states that this is a snippet, but with slightly different wording.) You want to use that snippet. Press the TAB key to insert **if** into your code. Then press the TAB key again to use the **if** snippet. (If you clicked somewhere else and your **IntelliSense** window disappeared, backspace over the **i** and retype it, and the**IntelliSense** window opens again.)

Visual C# code



1. Next, you use IntelliSense to enter more code to open an **Open File** dialog box. If the user clicked the **OK**button, the PictureBox loads the file that the user selected. The following steps show how to enter the code, and although it's numerous steps, it's just a few keystrokes:
   1. Start with the selected text **true** in the snippet. Type **op** to overwrite it. (In Visual Basic, you start with an initial cap, so type **Op**.)
   2. The **IntelliSense** window opens and displays **openFileDialog1**. Press TAB to select it. (In Visual Basic, it starts with an initial cap, so you see **OpenFileDialog1**. Ensure that **OpenFileDialog1** is selected.)
   3. Type a period (**.**) (Many programmers call this a dot.) Because you typed a dot right after**openFileDialog1**, an **IntelliSense** window opens, filled in with all of the **OpenFileDialog** component's properties and methods. These are the same properties that appear in the **Properties** window when you click it in Windows Forms Designer. There are also methods that can tell the component to do things (like open a dialog box).

|  |
| --- |
| **NoteNote** |
| The **IntelliSense** window can show you both properties and methods. To determine what is being shown, look at the icon to the left. You see a picture of a block next to each method, and a picture of a hand next to each property. There's also a lightning bolt icon next to each event. These pictures display as follows. |

* 1. Method icon
  2. Method icon
  3. Property icon
  4. Property icon
  5. Event icon
  6. Event icon
  7. Start to type **ShowDialog** (capitalization is unimportant to IntelliSense). The **ShowDialog()** method will show the **Open File** dialog box. After the window has highlighted **ShowDialog**, press TAB.
  8. When you use a method on a control or a component (referred to as *calling a method*), you need to add parentheses. So enter opening and closing parentheses: **()**

|  |
| --- |
| **NoteNote** |
| Methods are an important part of any program, and this tutorial has shown several ways to use methods. You can call a component's method to tell it to do something, like how you called the**OpenFileDialog** component's **ShowDialog()** method. You can create your own methods to make your program do things, like the one you're building now, called the **showButton\_Click()** method, which opens a dialog box and a picture when a user clicks a button. |

* 1. For Visual C#, add a space, and then add two equal signs (**==**). For Visual Basic, add a space, and then use a single equal sign (**=**). (Visual C# and Visual Basic use different equality operators.)
  2. Add another space. As soon as you do, another **IntelliSense** window opens. Start to type **DialogResult**and press TAB to add it.

|  |
| --- |
| **NoteNote** |
| When you write code to call a method, sometimes it returns a value. In this case, the**OpenFileDialog** component's **ShowDialog()** method returns a DialogResult value. DialogResult is a special value that tells you what happened in a dialog box. An **OpenFileDialog** component can result in the user clicking **OK** or **Cancel**, so its **ShowDialog()** method returns either DialogResult.OK or DialogResult.Cancel. |

* 1. Type a dot to open the DialogResult value **IntelliSense** window. Enter the letter **O** and press TAB to insert **OK**.

|  |
| --- |
| **NoteNote** |
| The first line of code should be complete. For Visual C#, it should be the following.  if (openFileDialog1.ShowDialog() == DialogResult.OK)  For Visual Basic, it should be the following.  If OpenFileDialog1.ShowDialog() = DialogResult.OK Then |

* 1. Now add one more line of code. You can type it (or copy and paste it), but consider using IntelliSense to add it. The more familiar you are with IntelliSense, the more quickly you can write your own code. Your final **showButton\_Click()** method looks like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492131(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

private void showButton\_Click(object sender, EventArgs e)

{

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

pictureBox1.Load(openFileDialog1.FileName);

}

}

* When you double-clicked the **Show a picture** button in Windows Forms Designer, the IDE automatically added a *method* to your program's code.
* Methods are how you organize your code: It's how your code is grouped together.
* Most of the time, a method does a small number of things in a specific order, like how your**showButton\_Click()** method shows a dialog box and then loads a picture.
* A method is made up of *statements*. Think of a method as a way to bundle statements together.
* When a method is executed, or *called*, the statements in the method are executed in order, one after another, starting with the first one.

The following is an example of a statement.

C#

pictureBox1.Load(openFileDialog1.FileName);

VB

pictureBox1.Load(openFileDialog1.FileName)

Statements are what make your programs do things. In Visual C#, a statement always ends in a semicolon. In Visual Basic, the end of a line is the end of a statement. (No semicolon is needed in Visual Basic.) The preceding statement tells your **PictureBox** control to load the file that the user selected with the**OpenFileDialog** component.

link to videoFor a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 5](http://go.microsoft.com/fwlink/?LinkId=205216) or [Tutorial 1: Create a Picture Viewer in C# - Video 5](http://go.microsoft.com/fwlink/?LinkId=205206).

You next add a comment to your code. A comment is a note that doesn't change the way the program behaves. It makes it easier for someone to understand what your code does. In Visual C#, two forward slashes (//) mark a line as a comment. In Visual Basic, a single quotation mark (') is used to mark a line as a comment.

After you add a comment, you test your program. You just built something that works, and although it's not done yet, it can already load a picture.

To add comments

1. Add the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492177(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

private void showButton\_Click(object sender, EventArgs e)

{

// Show the Open File dialog. If the user clicks OK, load the

// picture that the user chose.

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

pictureBox1.Load(openFileDialog1.FileName);

}

}

|  |
| --- |
| **NoteNote** |
| Your **showButton** button's Click event handler is now finished, and it works. You have started writing code, starting with an **if** statement. An **if** statement is how you tell your program, "Check this one thing, and if it's true, do these actions." In this case, you tell your program to open the **Open File** dialog box, and if the user selects a file and clicks the **OK** button, load that file in the PictureBox. |
| **NoteNote** |
| The IDE is built to make it easy for you to write code, and *code snippets* are one way it does that. A snippet is a shortcut that gets expanded into a small block of code.  You can see all of the snippets by selecting **Code Snippets Manager** from the **Tools** menu. The **if** snippet is in **Code Patterns**, inside the **Conditionals and Loops** subfolder. You can use this manager to browse existing snippets or add your own snippets.  To activate a snippet when typing code, type it and press the TAB key. Many snippets appear in the**IntelliSense** window, which is why you press TAB twice: first to select the snippet from the **IntelliSense**window, and then to tell the IDE to use the snippet. (IntelliSense supports the **if** snippet, but not the **ifelse**snippet.) |

1. Before you run your program, save your program by clicking the **Save All** toolbar button, which appears as follows.

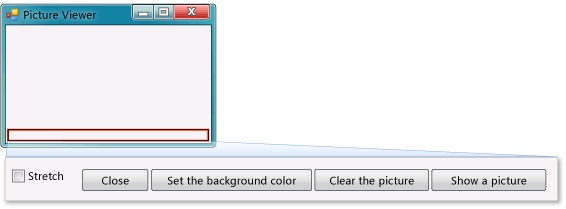
Save All button

Save All toolbar button

Alternatively, to save your program, from the **File** menu, click **Save All**. It's a best practice to save early and often.

When it's running, your program should look like the following picture.

Picture Viewer



To test your program

1. Press the F5 key or click the **Start Debugging** toolbar button.
2. Click the **Show a picture** button to run the code you just wrote. First, the program opens an **Open File** dialog box. Verify that your filters appear in the **Files of type** drop-down list at the bottom of the dialog box. Then navigate to a picture and open it. You can usually find sample pictures that ship with the Windows operating system in your **My Documents** folder, inside the **My Pictures\Sample Pictures** folder.
3. Load a picture, and it appears in your PictureBox. Then try resizing your form. Because you have your PictureBox docked inside a TableLayoutPanel, which itself is docked inside the form, your picture area will resize itself so that it's as wide as the form, and fills the top 90 percent of the form. That's why you used the TableLayoutPanel and FlowLayoutPanel containers: They keep your form sized correctly when the user resizes it.

To write code for additional buttons and a check box

* Add the following code.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492133(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

private void clearButton\_Click(object sender, EventArgs e)

{

// Clear the picture.

pictureBox1.Image = null;

}

private void backgroundButton\_Click(object sender, EventArgs e)

{

// Show the color dialog box. If the user clicks OK, change the

// PictureBox control's background to the color the user chose.

if (colorDialog1.ShowDialog() == DialogResult.OK)

pictureBox1.BackColor = colorDialog1.Color;

}

private void closeButton\_Click(object sender, EventArgs e)

{

// Close the form.

this.Close();

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

// If the user selects the Stretch check box,

// change the PictureBox's

// SizeMode property to "Stretch". If the user clears

// the check box, change it to "Normal".

if (checkBox1.Checked)

pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;

else

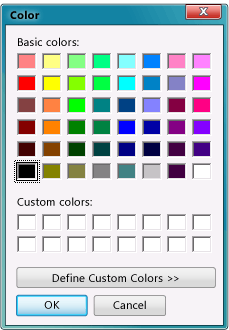
pictureBox1.SizeMode = PictureBoxSizeMode.Normal;

}

To run your program and set the background color

1. Press F5 or click the **Start Debugging** toolbar button.
2. Before you open a picture, click the **Set the background color** button. The **Color** dialog box opens.

Color dialog box



1. Select a color to set the PictureBox background color. Look closely at the **backgroundButton\_Click()** method to understand how it works.

|  |
| --- |
| **NoteNote** |
| You can load a picture from the Internet by pasting its URL into the **Open File** dialog box. Try to find an image with a transparent background, so your background color shows. |

1. Click the **Clear the picture** button to make sure it clears. Then, exit the program by clicking the **Close** button.

To try other features

* Change the color of the form and the buttons using the **BackColor** property.
* Customize your buttons and check box using the **Font** and **ForeColor** properties.
* Change your form's **FormBorderStyle** and **ControlBox** properties.
* Use your form's **AcceptButton** and **CancelButton** properties so that buttons are automatically clicked when the user presses the ENTER or ESC key. Make the program open the **Open File** dialog box when the user presses ENTER and close the box when the user presses ESC.

To continue or review

* To go to the next tutorial, see [Tutorial 2: Create a Maze](http://msdn.microsoft.com/en-us/library/dd492162(v=vs.100).aspx).
* To return to the previous tutorial step, see [Step 10: Write Code for Additional Buttons and a Check Box](http://msdn.microsoft.com/en-us/library/dd492133(v=vs.100).aspx).