

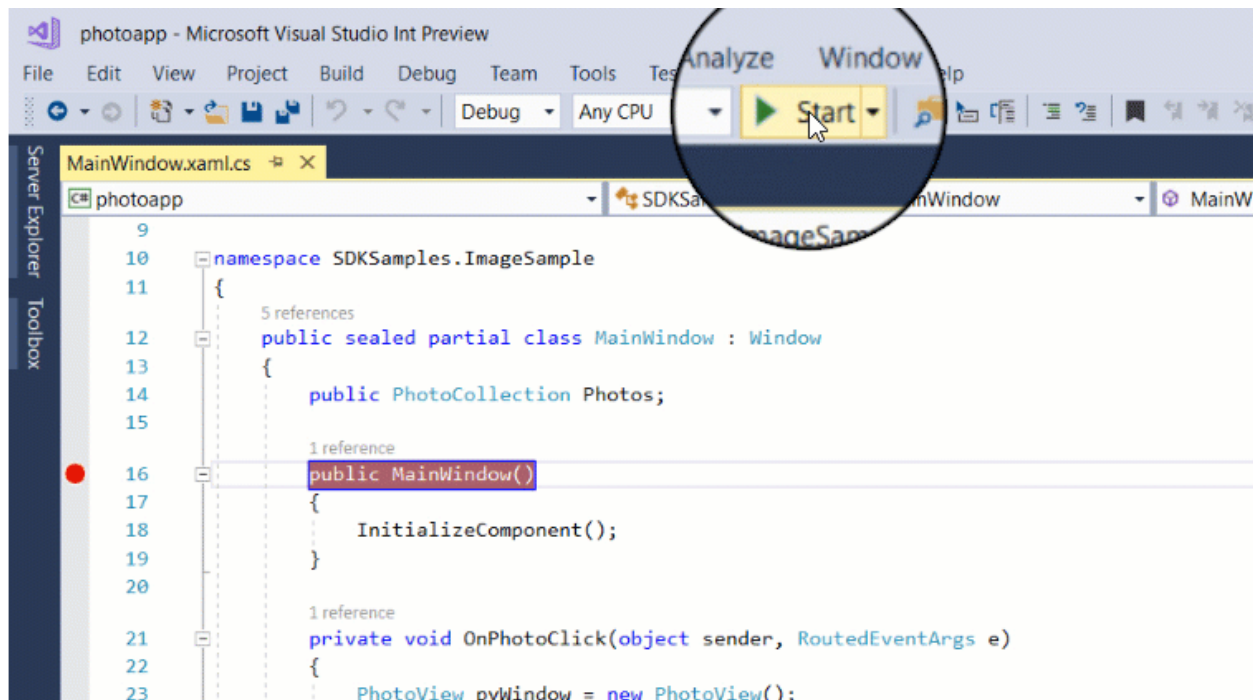
Debugging with Visual Studio

To debug, you need to start your app with the debugger attached to the app process. **F5 (Debug > Start Debugging)** is the most common way to do that. However, right now you may not have set any breakpoints to examine your app code, so we will do that first and then start debugging.

Set a breakpoint and start the debugger

Breakpoints are the most basic and essential feature of reliable debugging. A breakpoint indicates where Visual Studio should suspend your running code so you can take a look at the values of variables, or the behaviour of memory, or whether or not a branch of code is getting run.

If you have a file open in the code editor, you can set a breakpoint by clicking in the margin to the left of a line of code.



Start Debugging in the Debug Toolbar, and the debugger runs to the first breakpoint that it encounters. If the app is not yet running, F5 starts the debugger and stops at the first breakpoint. Breakpoints are a useful feature when you know the line of code or the section of code that you want to examine in detail.

Navigate code in the debugger using step commands

To start your app with the debugger attached, press **F11 (Debug > Step Into)**. **F11** is the Step Into command and advances the app execution one statement at a time. When you start the app with **F11**, the debugger breaks on the first statement that gets executed.

The yellow arrow represents the statement on which the debugger paused, which also suspends app execution at the same point (this statement has not yet executed).

```
5 references
12 public sealed partial class MainWindow : Window
13 {
14     public PhotoCollection Photos;
15
16     1 reference
17     public MainWindow()
18     {
19         InitializeComponent(); < 1ms elapsed
20     }
```

Step over code to skip functions

When you are on a line of code that is a function or method call, you can press **F10 (Debug > Step Over)** instead of **F11**. **F10** advances the debugger without stepping into functions or methods in your app code (the code still executes). By pressing **F10**, you can skip over code that you're not interested in. This way, you can quickly get to code that you are more interested in.

Step into a property

By default the debugger skips over managed properties and fields, but the **Step Into Specific** command allows you to override this behaviour. **Right-click on a property or field** and choose **Step Into Specific**, then choose one of the available options.

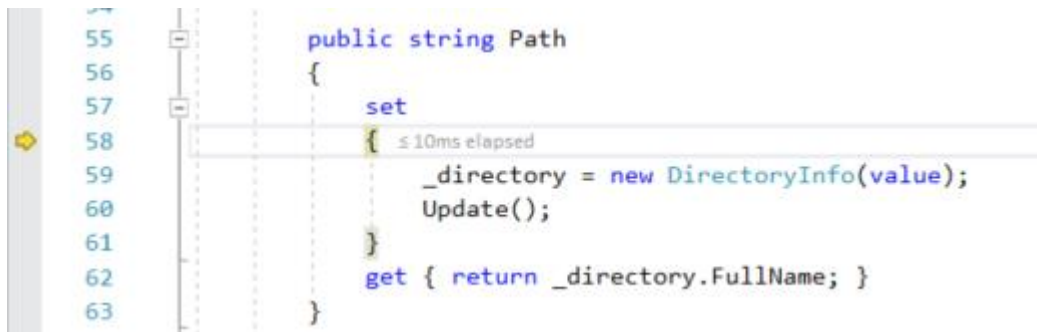
```
13 MainWindow mainWindow = new MainWindow();
14 mainWindow.Show();
15 mainWindow.Photos = (PhotoCollection)(this.Resources["Photos"] as Obj
16 mainWindow.Photos.Path = Environment.CurrentDirectory + "\\images";
17 }
18 }
19 }
```

Command	Shortcut
View Designer	Shift+F7
Quick Actions and Refactorings...	Ctrl+.
Rename...	Ctrl+R, Ctrl+R
Remove and Sort Usings	Ctrl+R, Ctrl+G
Peek Definition	Alt+F12
Go To Definition	F12
Go To Implementation	Ctrl+F12
Find All References	Shift+F12
View Call Hierarchy	Ctrl+K, Ctrl+T
Step Into Specific	
Run To Cursor	Ctrl+F10
Set Next Statement	Ctrl+Shift+F10
Go To Disassembly	Alt+G

Watch 1


- System.Environment.CurrentDirectory.get
- string.Concat
- DKSamples.ImageSample.PhotoCollection.Path.set**

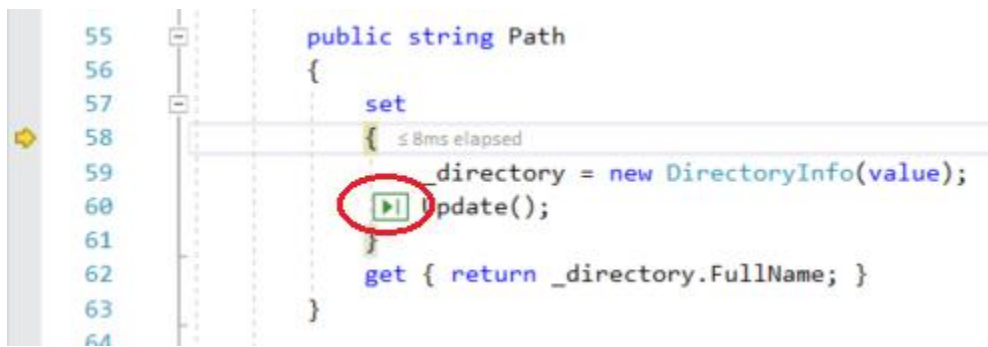
In this example, **Step Into Specific** gets us to the code for **Path.set**.



```
55 public string Path
56 {
57     set
58     {
59         _directory = new DirectoryInfo(value);
60         Update();
61     }
62     get { return _directory.FullName; }
63 }
```

Run to a point in your code quickly using the mouse

While in the debugger, hover over a line of code until the **Run to Click** (Run execution to here) button  appears on the left.



```
55 public string Path
56 {
57     set
58     {
59         _directory = new DirectoryInfo(value);
60         Update();
61     }
62     get { return _directory.FullName; }
63 }
```

Click the **Run to Click** (Run execution to here) button. The debugger advances to the line of code where you clicked. Using this button is similar to setting a temporary breakpoint. This command is also handy for getting around quickly within a visible region of app code. You can use **Run to Click** in any open file.

Advance the debugger out of the current function

Sometimes, you might want to continue your debugging session but advance the debugger all the way through the current function. Press **Shift + F11** (or **Debug > Step Out**). This command resumes app execution (and advances the debugger) until the current function returns.

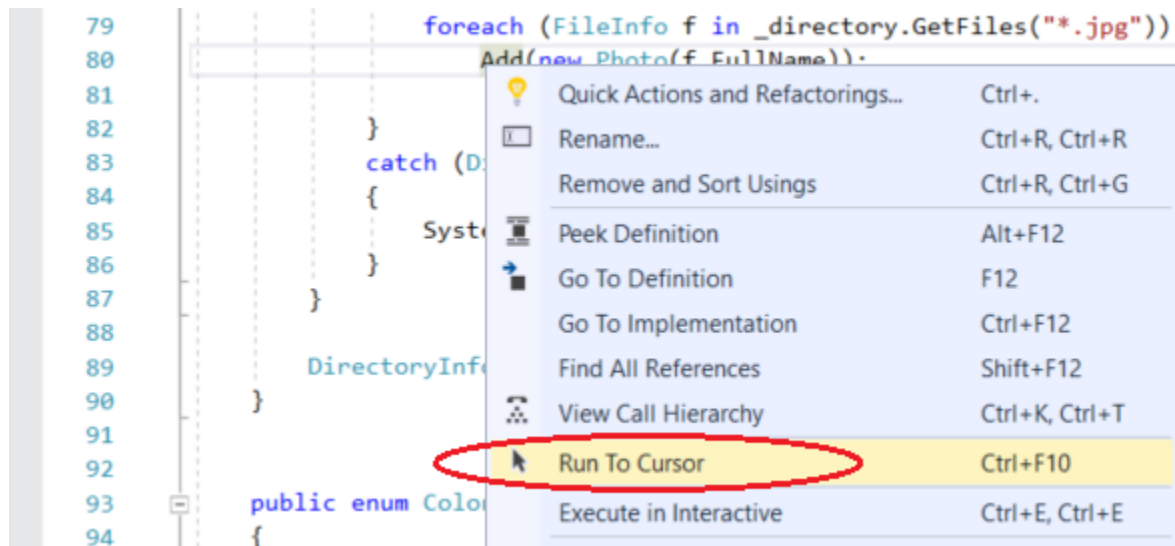
Run to cursor

When you are editing code (rather than paused in the debugger), right-click a line of code in your app and choose **Run to Cursor**. This command starts debugging and sets a temporary breakpoint on the current line of code.


If you have set breakpoints, the debugger pauses on the first breakpoint that it hits.


Press **F5** until you reach the line of code where you selected **Run to Cursor**.

This command is useful when you are editing code and want to quickly set a temporary breakpoint and start the debugger at the same time.



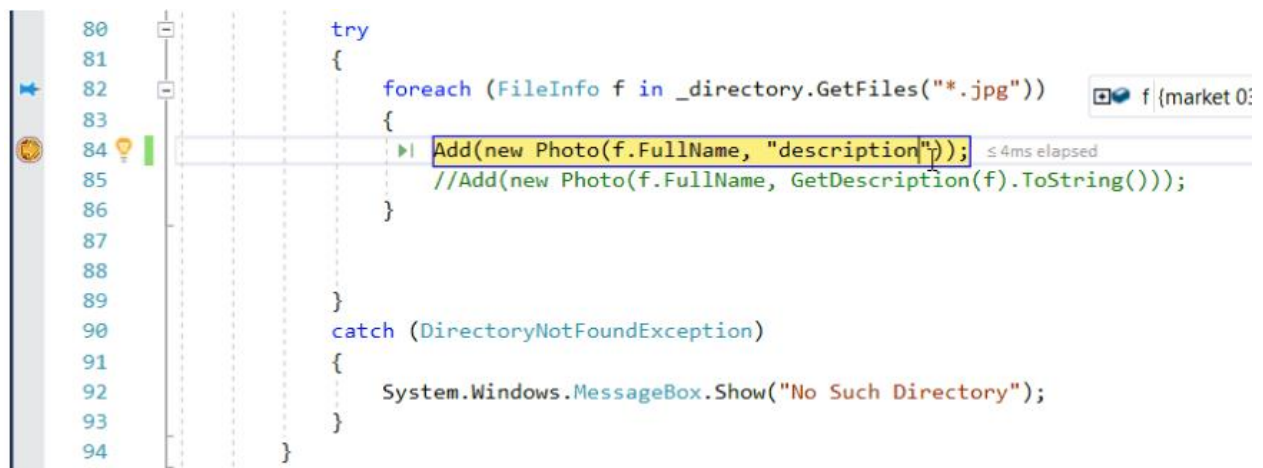
Restart your app quickly

Click the **Restart**  button in the Debug Toolbar (**Ctrl + Shift + F5**).

When you press **Restart**, it saves time versus stopping the app and restarting the debugger. The debugger pauses at the first breakpoint that is hit by executing code. If you do want to stop the debugger and get back into the code editor, you can press the red stop  button instead of **Restart**.

Edit your code and continue debugging (C#, VB, C++, XAML)

In most languages supported by Visual Studio, you can edit your code in the middle of a debugging session and continue debugging. To use this feature, **click into your code with your cursor** while paused in the debugger, **make edits**, and **press F5, F10, or F11** to continue debugging.



```

80     try
81     {
82         foreach (FileInfo f in _directory.GetFiles("*.jpg"))
83         {
84             Add(new Photo(f.FullName, "some new description"));
85             //Add(new Photo(f.FullName, GetDescription(f).ToString()));
86         }
87     }
88
89     }
90     catch (DirectoryNotFoundException)
91     {
92         System.Windows.MessageBox.Show("No Such Directory");
93     }
94 }

```

Press F10 Here!

Inspect variables with data tips

Start inspecting your app state (variables) with the debugger. Often, when you try to debug an issue, you are attempting to find out whether variables are storing the values that you expect them to have in a particular app state.

While paused in the debugger, hover over an object with the mouse and you see its default property value (in this example, the file name market 031.jpg is the default property value).

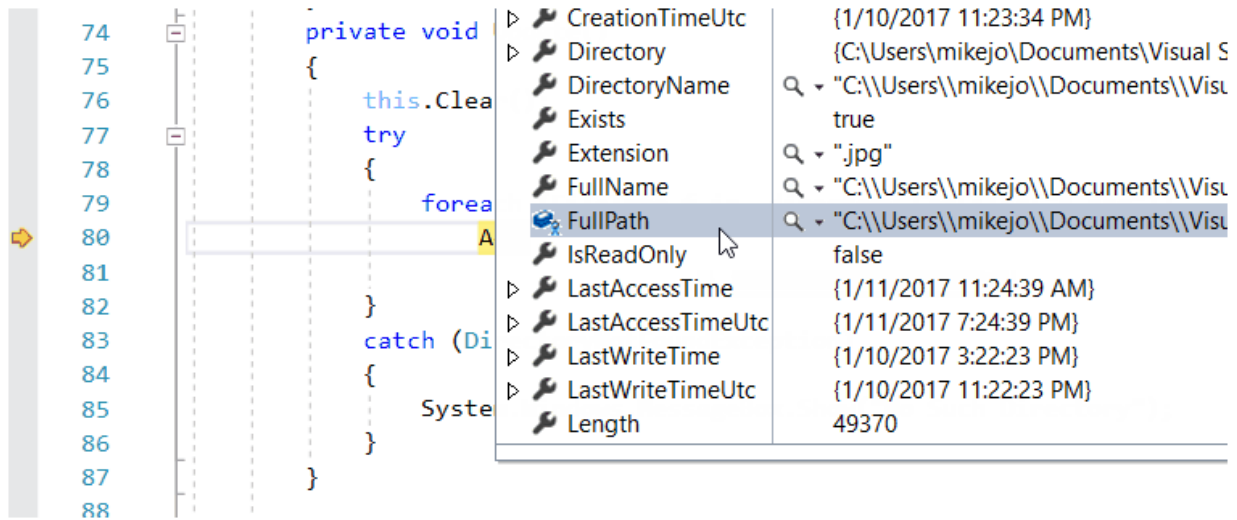
```

74     private void Update()
75     {
76         this.Clear();
77         try
78         {
79             foreach (FileInfo f in _directory.GetFiles("*.jpg"))
80                 Add(new Photo(f.FullName));
81         }
82         catch (DirectoryNotFoundException)
83         {
84             System.Windows.MessageBox.Show("No Such Directory");
85         }
86     }
87 }
88

```

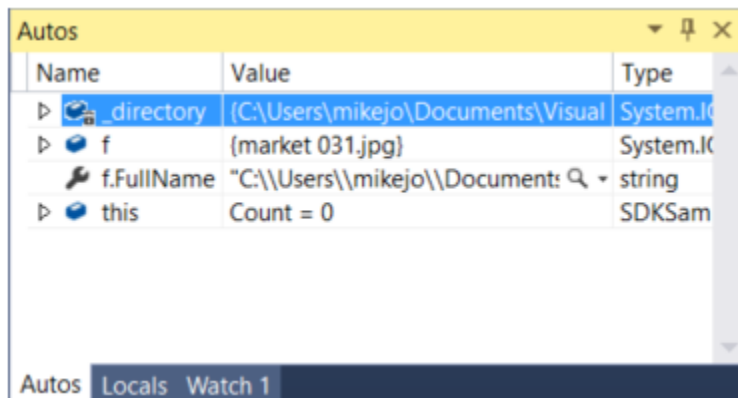
Expand the object to see all its properties (such as the FullPath property in this example).

Often, when debugging, you want a quick way to check property values on objects, and the data tips are a good way to do it.

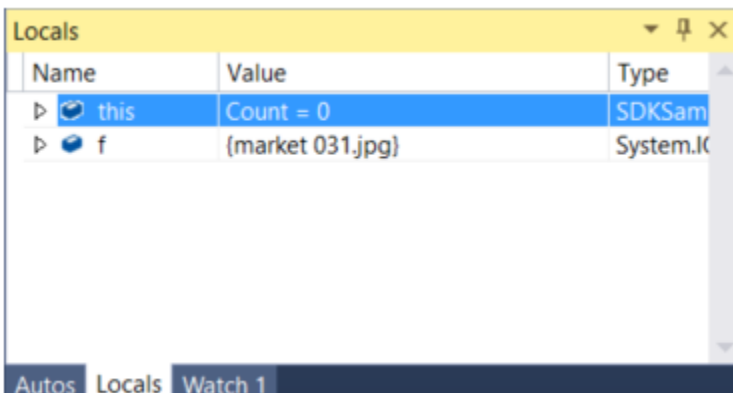


Inspect variables with the Autos and Locals windows

While debugging, look at the **Autos** window at the bottom of the code editor. In the **Autos** window, you see variables along with their current value and their type. The **Autos** window shows all variables used on the current line or the preceding line.



Next, look at the **Locals** window. The **Locals** window shows you the variables that are currently in scope.



Set a watch

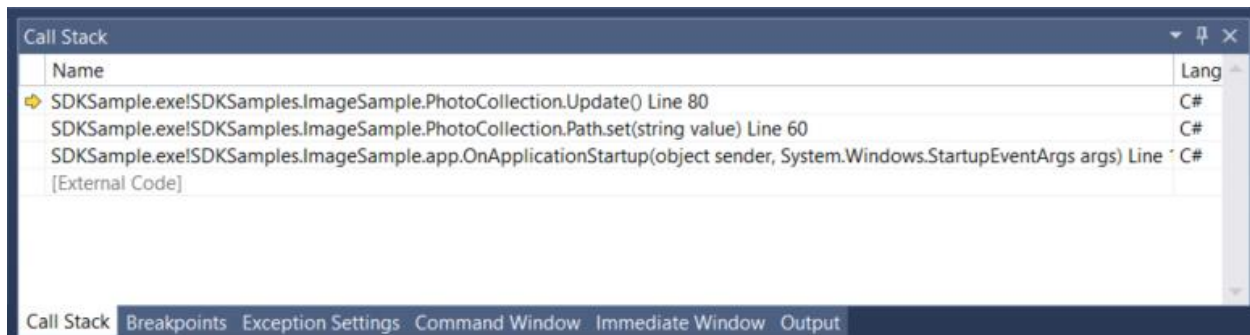
You can use a **Watch** window to specify a variable (or an expression) that you want to keep an eye on. While debugging, right-click an object and choose **Add Watch**.



In this example, we have a watch set on the **f** object, and you can see its value change as you move through the debugger. Unlike the other variable windows, the **Watch** windows always show the variables that you are watching (they're grayed out when out of scope).

Examine the call stack

Click the **Call Stack** window while you are debugging, which is by default open in the lower right pane.



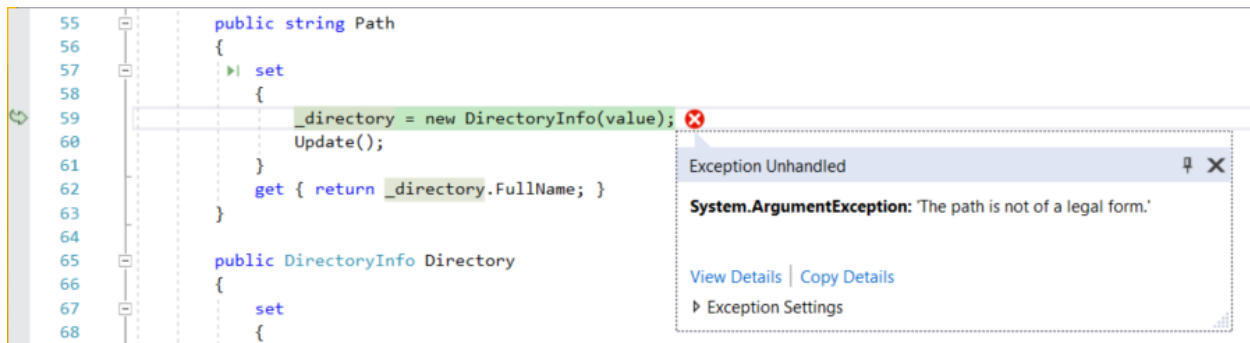
The **Call Stack** window shows the order in which methods and functions are getting called. The top line shows the current function (the Update method in this example). The second line shows that Update was called from the Path.set property, and so on. The call stack is a good way to examine and understand the execution flow of an app.

You can double-click a line of code to go look at that source code and that also changes the current scope being inspected by the debugger. This does not advance the debugger.

You can also use right-click menus from the **Call Stack** window to do other things. For example, you can insert breakpoints into specific functions, restart your app using **Run to Cursor**, and to go examine source code.

Examine an exception

When your app throws an exception, the debugger takes you to the line of code that threw the exception.



```
55 public string Path
56 {
57     set
58     {
59         _directory = new DirectoryInfo(value);
60         Update();
61     }
62     get { return _directory.FullName; }
63 }
64
65 public DirectoryInfo Directory
66 {
67     set
68     {
```

In this example, the **Exception Helper** shows you a `System.Argument` exception and an error message that says that the path is not a legal form. So, we know the error occurred on a method or function argument.

In this example, the `DirectoryInfo` call gave the error on the empty string stored in the value variable.

The Exception Helper is a great feature that can help you debug errors. You can also do things like view error details and add a watch from the Exception Helper. Or, if needed, you can change conditions for throwing the particular exception. Expand the **Exception Settings** node to see more options on how to handle this exception type.