

Programming Logic - Beginning

152-101

Debugging Applications

Notes	Activity
-------	----------

Quick Links & Text References

- [Debugging Concepts](#) Pages
- [Debugging Terminology](#) Pages
- [Debugging in Visual Studio](#) Pages
 - [Breakpoints](#) Pages
 - [Watches](#) Pages
 - [Stepping](#) Pages
 - [Temporary Messages](#) Pages
 - [Debug.WriteLine](#) Pages

Debugging

- Debugging is removing logic and runtime errors from programs
 - These are the most difficult errors to uncover
 - All Integrated Development Environments include tools called *debuggers* that help you uncover these errors
- These tools allow you to pause a program and then execute statements one at a time.
 - By executing statements one at a time, you can see the order the program is executing statements
 - When you are stopped, you can examine variables' content and property values.

Debugging Terminology


- Breakpoints
 - Breakpoints allow you to designate a line (or lines) of code where the program should stop (and enter debugging mode).
 - Allows you to skip code you know is functioning properly and stop the program where you feel the error is occurring
 - You can also set conditional breakpoints (less used) that only stop the program if a certain condition exists

Notes	Activity
<ul style="list-style-type: none">• Watches<ul style="list-style-type: none">➤ Watches allow to <i>watch</i> the contents of selected variables➤ Most of today's debuggers also allow you <i>touch</i> (point to with mouse) a variable to see its contents instead of (or in addition to) setting watches.• Step Over, Step Into, Step Out<ul style="list-style-type: none">➤ Once a program is in debug mode you can tell the program to execute the next statement➤ If the next statement includes a method call, Step Over allows you skip over the method's statements<ul style="list-style-type: none">– When stepping through a program, you'll normally use Step Over.– I only use Step Into to debug methods that I've written (not built-in methods)➤ If the next statement includes a method call, Step Into allows you to transfer program control to the first line in the method<ul style="list-style-type: none">– If Step Over causes an error, run the program again and choose Step Into instead➤ If you inadvertently step into a method, Step Out allows you to return program control to the statement that called the method.<ul style="list-style-type: none">– The remaining statements in the method are executed, but they are not debugged (no stops)• Displaying Temporary Messages<ul style="list-style-type: none">➤ Instead of using the debugger, you can also choose to write statements that cause values or messages to appear while the program is running➤ Be sure to remove these displays or at least comment them out once you completed debugging.	

Notes

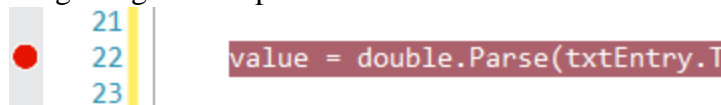
Activity

Debugging in Visual Studio

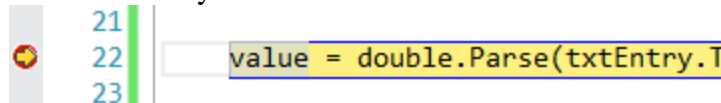
- To start Debugging, you can
 - click the  **Start** button on the toolbar; or
 - select Debug menu, Start Debugging; or
 - Press F5.
 - In Visual Studio, debugging is really not much different than running a program. You'll enter debug mode when the first breakpoint is reached

- Setting Breakpoints

- Click on the far left side of the line where you want to set the break (inside the gray area)
- Or, put your cursor on the line of code you want to stop on, click Debug, click Toggle Breakpoint.
- A red circle will appear in the gray area and the line itself will appear with a dark red background designating a breakpoint has been set on this line.



- When the program is executing and reaches the breakpoint, it pauses and is in *Break* mode.
- The line of code you stop at will turn yellow. This is the line of code that *will be executed next*. It has not been executed yet.

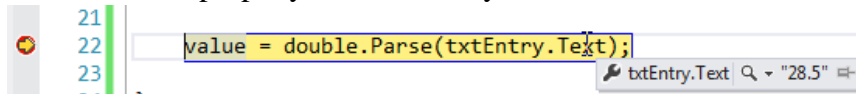


- One common misperception is that the highlighted line has already been executed. **THIS IS NOT TRUE.**
 - The highlighted line is the **next** line to be executed
- You can **remove the break point** by
 - clicking the red breakpoint dot or
 - clicking on the line and clicking Breakpoint ▸ Delete in the popup menu

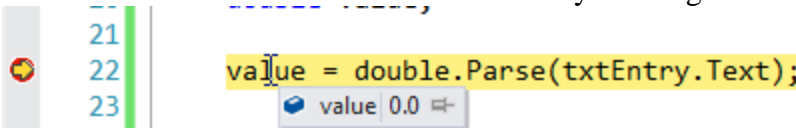
Notes	Activity
-------	----------

- Setting Watches

- While in *break* mode, Visual Studio allows you to examine variables and values stored in properties.
- Using your mouse, point to (don't click) the variable or property whose value you wish to see



- In the example above, the mouse pointer (I) is resting on the word *Text*. Remember, you must be in break mode to view variables by touching them.






- In this example, the mouse pointer is resting on the variable named *value*
 - Note: **because this line has not yet been executed**, the variable *value* has a value of zero even though *txtEntry.Text* has a value of “28.5”
 - After you step past this line, the command will execute and value will be assigned
- **Tip:** If the object you're pointing to doesn't display a value, try highlighting the entire object name (drag over with mouse) and then touch it the mouse pointer.

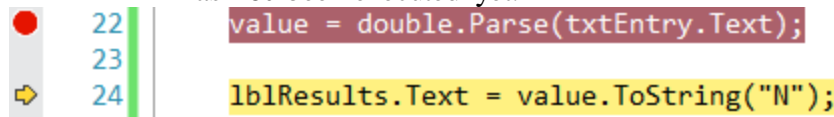
Notes

Activity

- Because you can see the values of variables simply by touching them, I don't normally set watches. Visual Studio however, does allow you set variable watches. I'll describe what I feel is the easiest way to set these watches.
 - Note: you must be in Debug **break** mode to set watches
 - Select (drag with mouse) the variable you want to set a watch for
 - Right-click the selected item
 - Choose Add Watch from the popup menu
- | Name | Value |
|---------------|--------|
| value | 0.0 |
| txtEntry.Text | "28.5" |
- A Watch window will appear at the bottom of the IDE that displays all the variables you've set watches for and their current value.
 - If the variable is out of scope, no value will appear
 - If the Watch window isn't visible, click Debug ▸ Windows ▸ Watch ▸ Watch 1
 - You can also examine variables using the *Autos* and *Locals* windows (Debug ▸ Windows).
 - *Autos*: displays a list of variables and objects appearing in the current statement as well as the 3 statements before and 3 statements after the current statement.
 - *Locals*: displays a list of all variables in the current procedure. However, form objects (like txtEntry) don't appear
 - **Tip**: if these windows are not visible, open the Debug ▸ Windows menu and click the window you want to see

Notes	Activity
-------	----------

- Stepping Through the Program
 - Visual Studio provides convenient shortcut keys and toolbar buttons for stepping through a program in debug mode
 - Step Over: **F10** 
 - Step Into: **F11** 
 - Step Out: **Shift-F11** 
 - These commands are also available under the Debug menu (note shortcut keys are listed)
 - If your shortcut keys are different (F8, F9, Shift-F8), you can switch them to our default keys
 - Tools ▶ Options
 - Expand the *Environment* group and then choose *Keyboard*
 - Change the first combo box (*Apply the following...*) to (**Default**)
 - Alternatively, you can leave them the way they are and just remember these shortcut keys or use the toolbar buttons
 - When you step through your program, **the next line to be executed** is highlighted in yellow and a yellow arrow is displayed in the gray margin
 - In the example below, the programmer stopped the program using a breakpoint on line 22. The programmer then stepped over (F10) line 22. Line 24 is now the **next** line to be executed—it has **not** been executed yet.



- Displaying Temporary Messages
 - Visual Studio provides numerous ways to temporarily display values while you are debugging your program
 - Temporarily add a label to your form and display debug values there
 - Use a Message Box to temporarily display values
 - See the [Message Box](#) notes

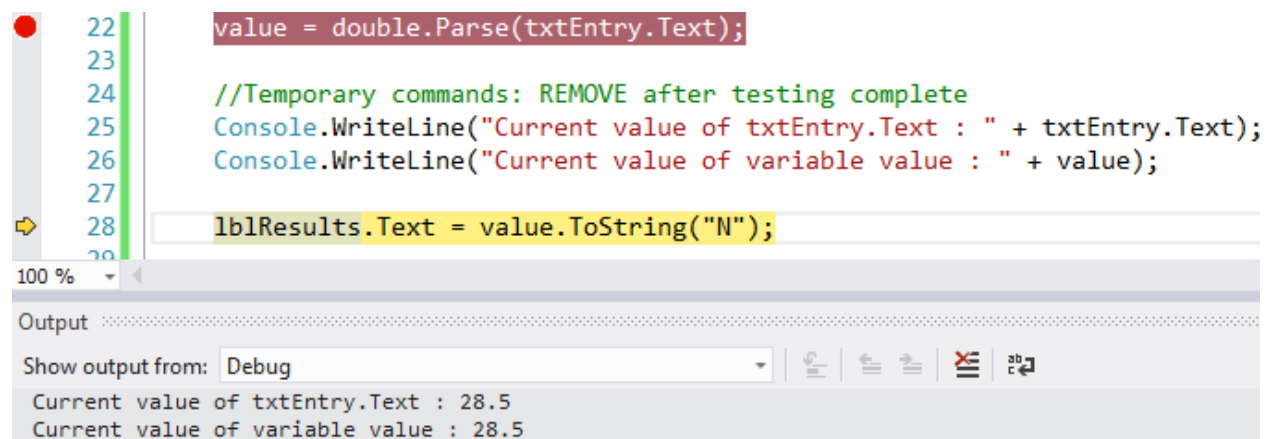
Notes	Activity
-------	----------

- My favorite technique is to use the `Console.WriteLine` command
 - The `Console.WriteLine` method provides the ability to see program values in the *Output Window*, as the program is running.
 - You do not need to stop the program to see the values, you issue the following statement and you can display values or monitor the program flow.
 - Syntax:
`Console.WriteLine (message/string)`
 - Message/string can be a single value or you can concatenate strings with numbers.

```
Console.WriteLine("Value of amount is " + amount);
```

- If your *Output* window doesn't display, use the **Debug** ▶ **Windows** menu to display it.
- The following commands in the `Console` class might also be useful
`Write` (no automatic carriage return)
`WriteIf`
`WriteLineIf`

Debug.WriteLine Example




The screenshot shows a Visual Studio window with a code editor and an output window. The code editor displays the following code:

```
22 value = double.Parse(txtEntry.Text);  
23  
24 //Temporary commands: REMOVE after testing complete  
25 Console.WriteLine("Current value of txtEntry.Text : " + txtEntry.Text);  
26 Console.WriteLine("Current value of variable value : " + value);  
27  
28 lblResults.Text = value.ToString("N");
```

The output window, titled "Output", shows the following text:

```
Current value of txtEntry.Text : 28.5  
Current value of variable value : 28.5
```

Notes	Activity
<ul style="list-style-type: none">➤ In the example above, the programmer set a breakpoint at line 22. When the program stopped, the programmer stepped through the program (F10) until line 28 was highlighted.➤ The Console.WriteLine methods on lines 25 and 26 caused the messages to display in the Output Window➤ The program was put into debug mode to help with the explanation of the Console.WriteLine commands. The programmer could remove the breakpoint on line 22 and run the program again. Even without the breakpoints, without entering debug mode, the program will still display the messages in the Output Window ➤ The Console.WriteLine method only displays the output in the Output Window. The Output Window is only available when using the IDE. Executable versions of programs ignore the Console.WriteLine commands.<ul style="list-style-type: none">– Executable versions of your programs are the ones you deliver to users (customers). These executable versions of programs have no way to display the Console.WriteLine messages. ➤ Once you've finished debugging your program, you can delete the Console.WriteLine commands<ul style="list-style-type: none">– Personally, I comment them out. Then, if I need to debug my program in future, I can easily reinstate them (remove commenting)– Optionally, because the executable version doesn't display the messages anyway, you could simply leave the Console.WriteLine methods in your code<ul style="list-style-type: none">▪ I still prefer to comment them out<ul style="list-style-type: none">♦ Too many Console.WriteLine commands can clutter your code, making it hard to maintain♦ Too many Console.WriteLine commands can cause a flood of messages in the Output Window, making it hard to find the messages applicable to your current debug problem.♦ At any time, you can right-click the Output Window and choose Clear All or click the Clear All toolbar button 	

Notes	Activity
-------	----------

- Advanced programs often create *log files* instead of (or in addition to) console messages.
 - These log files are text files written to by the program as it is running.
 - If the program crashes while the user is using it, the log files can be analyzed by programmers to determine what caused the program crash.
 - Log files can become quite large and users have the ability to delete them (unless they are well hidden).