Using Debugger with Visual Studio Community 2015

Updated: Amir Kamil
Fall 2015

Updated: ML Dorf
Winter 2014

Original: Anthony Sottile
Winter 2011
# Table of Contents

1. Introduction........................................................................................................... 3
2. Revision History .................................................................................................... 3
3. Starting the Debugger .............................................................................................. 4
4. Setting Breakpoints .................................................................................................. 5
5. Step Out, Step Into, Step Over .............................................................................. 7
   5.1 Step Over ........................................................................................................... 7
      5.1.1 Where to find Step Over? ......................................................................... 7
      5.1.2 Example Step Over .................................................................................. 8
   5.2 Step Into ............................................................................................................ 9
      5.2.1 Where to find Step Into? ......................................................................... 9
      5.2.2 Example Step Into .................................................................................. 10
   5.3 Step Out ............................................................................................................ 11
      5.3.1 Where to find Step Out? ......................................................................... 11
      5.3.2 Example using Step Out ......................................................................... 12
6. Locals ...................................................................................................................... 13
7. Watch ..................................................................................................................... 15
8. Conditional Breakpoints ......................................................................................... 17
1 Introduction

This document outlines debugging using Visual Studio Community 2015. It will start with basics and then delve into more advanced debugging techniques.

This document directly references menus and keyboard shortcuts from Visual Studio Community 2015. The concepts for Xcode follow the same paradigm, but obviously differ in keyboard shortcuts, icons, and menus.

This document assumes you are using Microsoft Visual and have created a C++ Project using “Empty Project” and have a program written. The example used is the trajectory computation program from Zyante 3.15, placed in a file named trajectory.cpp.

2 Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Author</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Anthony Sottile</td>
<td>Initial Draft using Visual Studio 2010</td>
</tr>
<tr>
<td>1.1</td>
<td>Anthony Sottile</td>
<td>Resized Images, Fixed Last Pagebreak</td>
</tr>
<tr>
<td>2.0</td>
<td>ML Dorf</td>
<td>Recaptured images for Visual Studio Pro 2013</td>
</tr>
<tr>
<td>3.0</td>
<td>Amir Kamil</td>
<td>Changed example and recaptured images for Visual Studio Community 2015</td>
</tr>
</tbody>
</table>
3 Starting the Debugger

Before debugging ensure your code builds.

This can be done by going to **Build ➔ Build Solution** or by pressing **F7**

Once your project builds you can begin debugging (running your program) by doing one of the following things:

**Going to Debug ➔ Start Debugging:**

Or by clicking the Green debug arrow:

Or by pressing **F5**
4 Setting Breakpoints

Before debugging it is often useful to set a breakpoint.

A breakpoint tells the debugger to pause execution so a developer can analyze what is going on in a part of their program. Breakpoints pause execution before running the line of code they are set on. Breakpoints are especially useful for figuring out exactly what is going on. Breakpoints are also useful for pausing execution after everything has finished in order to capture output, etc.

To set a breakpoint, all you need to do is click on the margin left of a line of code. Note that you can only set breakpoints on actual code and not on comments.

Place to click for breakpoint:
Start debugging. If your breakpoint is hit, you will see a yellow arrow over the breakpoint. Note that at this point, your program is still debugging/running, but the debugger has paused the execution. In the screenshot below, the program is about to execute line 26. **Note that the yellow arrow shows the line that is about to be executed.**
5 Step Out, Step Into, Step Over

Step Into, Step Over, and Step Out allow you to navigate execution after your breakpoint has been hit. You can use Step Over, Step Into, and Step Out while execution is paused so you can watch your code execute and utilize features such as Locals and Watch.

A brief description of Step Over, Step Into, and Step Out is provided here; for a more in-depth example, see below.

- **Step Over** – When execution is paused, the program executes the current line and the debugger automatically breaks at the next line.
- **Step Into** – When execution is paused at a function call, the function is called and the debugger automatically breaks at the beginning of the function which is called (if the function is within user code).
- **Step Out** – When execution is paused inside a function call, the function execution is completed and the debugger automatically breaks execution after the execution of the function.

5.1 Step Over

When execution is paused (at a breakpoint) you can use Step Over to execute the current line (where the yellow arrow is pointing) and automatically break at the next line.

5.1.1 Where to find Step Over?

One can perform a Step Over by going to **Debug ➔ Step Over**

Or by clicking this button:

Or by pressing **F10**
5.1.2 Example Step Over

Take the following breakpoint (execution arrow is at line 26):

```c
double DegToRad(double inDeg) {
    return ((inDeg * PI_CONST) / 180.0);
}

int main() {
    const double GRAVITY = 9.8; // Earth gravity (m/s^2)
    double launchAngle = 0.0; // Angle of launch (rad)
    double launchVelocity = 0.0; // Velocity (m/s)
    double elapsedTime = 1.0; // Time (s)
    double yLoc = -1.0; // Obj's location above ground

    // Code execution...
}
```

After pressing Step Over (execution is at line 27):

```c
double DegToRad(double inDeg) {
    return ((inDeg * PI_CONST) / 180.0);
}

int main() {
    const double GRAVITY = 9.8; // Earth gravity (m/s^2)
    double launchAngle = 0.0; // Angle of launch (rad)
    double launchVelocity = 0.0; // Velocity (m/s)
    double elapsedTime = 1.0; // Time (s)
    double yLoc = -1.0; // Obj's location above ground

    // Code execution...
}
```
5.2 Step Into

When execution is paused at a function call (at a breakpoint) one can use Step Into to look at the code being executed in the function. The debugger calls the function and automatically breaks execution at the beginning of the function call.

5.2.1 Where to find Step Into?

Step Into can be found at **Debug → Step Into**

Or by clicking this button:

Or by pressing **F11**
5.2.2 Example Step Into

When you want to see exactly how a function is executing, you will want Step Into. Take the following breakpoint (At line 36 launchAngle = DegToRad(launchAngle);):

```cpp
20    // Convert degree value to radians
21    double DegToRad(double inDeg) {
22        return ((inDeg * PI_CONST) / 180.0);
23    }
24
25    int main() {
26        const double GRAVITY = 9.8; // Earth gravity (m/s^2)
27        double launchAngle = 0.0; // Angle of launch (rad)
28        double launchVelocity = 6.0; // Velocity (m/s)
29        double elapsedTime = 1.0; // Time (s)
30
31        double yLoc = -1.0; // Obj’s location above ground
32        double xLoc = 0.0; // Obj’s horiz. loc. from star
33
34        cout << "Launch angle (deg): ";
35        cin >> launchAngle;
36        launchAngle = DegToRad(launchAngle); // To radians
37
38        cout << "Launch velocity (m/s): ";
```

After pressing Step Into the execution has called DegToRad and automatically paused at the beginning of the function call: (Line 21 inside DegToRad).
5.3  Step Out

When Execution is paused inside a function call, one can use Step Out to finish execution of that function call, and when that function returns to where it was called, the debugger will automatically pause execution.

5.3.1  Where to find Step Out?

Step Out can be found by going to Debug ➔ Step Out

Or by clicking this button:

Or by pressing \texttt{Shift + F11}
5.3.2 Example using Step Out

Assume execution is paused inside the DegToRad function call:

After performing Step Out, the function will finish executing (lines 21-23) and then automatically pause execution where it was called (in this case line 36).
6 Locals

At any time when execution is paused you can use Locals to see the local variables available to that line of code at that execution time. Locals is often very useful to watch the values of your variables as your program executes.

In Visual Studio, one can bring up the Locals window by going to Debug → Windows → Locals
Often the Locals window is at the bottom of the screen. Here is an example where execution is paused, showing the local variables at the bottom.
Watch is similar to Locals in that you can see what the values of variables are at certain times, except with Watches you can construct expressions to see their values. For example, if you have a variable $a$ and a variable $b$, and you wanted to know whether $a$ was equivalent to $b$ as you executed your program, you could set a watch for $a == b$ that would update as you stepped through your program.

In Visual Studio, one can bring up the Watch window by going to Debug $\rightarrow$ Windows $\rightarrow$ Watch $\rightarrow$ Watch 1. One can add a watch by typing in an expression under Name in the Watch window.
Often Watch is at the bottom of the screen. Here is an example showing paused execution and some sample expressions:
8 Conditional Breakpoints

A conditional breakpoint is a special kind of breakpoint that allows you to break your execution only when a special condition is met. For example, say in the example above I only wanted to break when `elapsedTime` was greater than 10.0. I can use a conditional breakpoint!

One can set a conditional breakpoint by right clicking on an existing breakpoint and selecting “Conditions…"

This will bring up a dialog where you can enter an expression:

In our case I set the expression to be when `elapsedTime > 10.0`. Execution will pass over this breakpoint unless this condition is met. This is often useful for identifying bugs with very strange cases!