Slide 1

Welcome I’m Philip and I’m going to go through some of the debugging features available in Visual Studio.

Debugging is something most people don’t like doing, but we still write code that doesn’t work first time. So let’s become better at debugging and spend less time doing it.

Slide 2

I’ve mentioned the previous languages I have used because in many ways people go about debugging in the same way as they did back historically in the 90s and before with COBOL.

Slide 3

These are the things I am going to go through.

You may discover a bug when you are developing, or you may have one reported to you, what do you?

Visual Studio Options

There are some setting in visual studio that are useful to know

Attributes that help you view information when debugging

Provide an easy way to view information about variables

Go through the windows that are available in Visual Studio under the debug menu item

Examining dumps – how to use dumps with visual studio

Symbol servers - what they are, and why you should have them

Visual Studio and links to browsers

Feature in the enterprise edition

Slide 4

So you find out there is a problem - what do you do.

I’ve seen it where people open visual studio and start pressing F10 or F11, Don’t - you may find the bug that way but it will be a long process

So firstly don’t panic

If the bug has been reported to you, arrange a schedule of when you will update the people concerned. You don’t want them coming to you every 10 minutes, and from their point of view they want to know something is happening.

Gather details about the bug, what happened - is it repeatable, how it happened.

Reproduce it in an environment you have control of, it maybe it only happens on a certain browser, on a certain operating system, or on a certain type of phone.

Have a think what possible might be happen, is there a certain area of code that seems likely.

Write a test to reproduce code – you don’t want to have to go through the whole scenario as you fix the bug. Maybe to help diagnose the bug you will even write a number of small programs with a very exact purpose.

Slide 5

Visual Studio options

There are two main areas where you have options when you are debugging

Options for debugging it self and options for exceptions.

Slide 6

Debugging Options

Either under the Debug menu item or via within tools options.

As you can see there are there are lots of options

Enable Just My Code - limits the debugging session to only your code, but sometimes you might want to be able to debug frameworks etc. Or dlls you have included

To distinguish user code from non-user code, Just My Code looks at three things: DBG Files, PDB files, and optimization.

From vs2013 this also applies to javascript and jquery

Other options include Show Run to click button, enable edit and continue

There will be a link at the end to explain all the options

Slide 7

Onto the exception options, - debug > windows > Exception settings

The exception options allows you to define what happens when an exception is thrown, for visual studio to pause and break you need the appropriate exception ticked. In the setting windows you can search for exceptions.

Slide 8

As shown by looking for divide, you can further define when an exception will break using conditions for example you can specify that an exception should not break when in a particular Module

Note the divide by zero exception does not fire for doubles you get an answer of infinity. In the early versions of 15.8 this was displayed as 8 in the watch/local windows.

Slide 9

Debugging Attributes. There are 3 kinds Display, Browsable and Type Proxy

Debugger display allows you to define what is going to be show in some of the windows when looking at at object

Slide 10

When looking at a list the default shows you the object type as seen

If we add a DebuggerDisplay attribute to the class we can specify what should be shown.

The example shows the Name property for each entry.

Slide 12

You can use functions but this will impact performance

Slide 13

DebuggerBrowsable allows you to define what is shown by default for a class - especially useful when you have a nested class

Slide 14

Collapse shows you at the class level

Slide 15

None doesn’t show the class at all

Slide 16

And Root Hidden, shows you the properties of the class without seeing the class level.

Collapse is the default.

Slide 17

This allows you to project you class through a proxy class. The proxy class is then seen in the watch window. Raw information still available

Slide 18

Debugger visualisers - When you are viewing the value of a property e.g. through the watch window or a datatip these allow you to see the data in a laid out format. There are a number build in, json, xml, string, html

There are also a large number in the the marketplace many free for example DataTable, there are also alternatives to the built in ones.

Slide 19

Or you can write one yourself, the concept is easy enough, the harder part is getting the format as you want it.

Slide 20

Datatips

Allow you to see the value of a variable in the code window. You can also pin the datatip so it is always visible when you are in that area. You can also add a comment to it, this is useful as you can make notes about that variable. Datatips are exportable, and importable, which means you can share your debugging progress, or hand it over. All through the debug menu.

Slide 21

The locals and autos windows show the values of variables the locals is for the method, the autos is variable used in the 2-3 lines around where you are that are in scope.

Slide 22

The call stack, this shows the nesting of methods in classes of where you are at the moment, if you click on one of the levels the locals’ window changes with it. You can add and remove data from the window by right clicking, same examples shown on the slide. This saves adding to the watch window.

Slide 23

The watch window allows you to view a variable, and for classes/structures expand and see the properties. When stepping into a method a variable might become out of scope and not have a value, you can still see the value, if you right click the variable before it goes out of scope and select make object id, it gets given an id, $1 the example in the slide. $1 then is visible when the variable is out of scope. It may also carry on showing a variable once the variable has died until it is garbage collected. This is useful when reference types are passed into methods. You can see if the reference variable has changed.

Slide 24

Multi thread and parallel processing can be a problem when debugging.

Slide 25

The thread window

This shows you what threads you have in your program. You can mark threads by flagging them, you can also search for particular threads using the search window, show different details by selecting what columns to show, and freeze and thaw threads. This is very useful when you are stepping through code and it is jumping from one thread to another.

If you have a number of threads in your application, you can come to this windows - select all the threads right click and freeze and then select the thread you are interested in and thaw that one, and then F10 will take you on that threads path.

Naming threads can help identify thread, this can be in the threads window by using rename, or programmable. The caveat here is that this may not work if you are using tasks or thread pool threads as threads can only be named once.

Slide 26

The debug toolbar allows you to see where threads are clicking the far right icon. This will show in the column on the left of the source - where breakpoints are shown, using the thread icon to shown where each thread is. By hovering on this icon using can see the thread id.

Slide 27

There is another toolbar debug location, if this is not visible can be displayed via the toolbar menu option. This toolbar allows you to switch processes/ threads and also select an item in the relevant call stack

Slide 28

There are windows to help with debugging parallel code as well. If we use the code on the slide as an example, just a simple loop that counts to 10 for each person.

Slide 29

We can use the parallel stack window to view the call stack for each of the parallel loops, here we can freeze or flag threads as well as select the call stack we want.

Slide 30

Also in the example there are 4 versions of i and name one for each person, using the parallel watch window we can see the value of each in thread, the variables are list horizontally.

Slide 31

Breakpoints

The bread and butter of a lot of debugging, these can also be exported and imported and labelled. Labelling is very helpful as you can use the label to give a purpose to it, and help you remember what you are using the breakpoint for.

Slide 32

Data breakpoints, in many of the older languages you could set a breakpoint on a variable when it changed value. That was done by watching a memory location, this is won’t work will .net and managed code. However we can set a breakpoint on the set method for a property, so maybe instead of fields use private properties and set breakpoints on the set.

Slide 33

You want to break when you return from a method but before anything else is done, e.g the value from the method is assigned to a variable in the calling method. The way to do this is to step into the method that you wish to break on return, look at the call stack

Slide 34

Breakpoints can be set in the call stack, so set a breakpoint on the calling method.

Slide 35

With Condition breakpoints you can get the breakpoint to only fire in certain conditions. E.g with this code when i =5;

Slide 36

If we set at breakpoint at the line we wish to break at and right click and select condition and now you can set the condition e.g i == 5; Now the breakpoint will only cause the code to break when i = 5. However this is expensive in time, as the code actually stops every time the breakpoint is hit, visual studio then does the comparison and then continues if the comparison is false. If the breakpoint is likely to be hit a lot, this can be a problem. You can just put a normal if statement around the breakpoint. Though you have to remember to take it out before going live. Which of course is forgotten

Slide 37

So if you do put an extra condition is and check that the debugger is attached.

Slide 38

Hit count = you can also have the breakpoint triggered when it is hit for the nth time. This is helpful when you have a method or line of code that throws an exception but not every time, it may be on the 1500th time or even unknown time.

If the number is unknown set the hit count very high, run the code.

Slide 39

When the exception is thrown - look at the breakpoint window and this will tell you what the hit count current is. Now change the hit count condition to that number and re run, now you will break just before the exception.

Slide 40

Filter - for use in mutli threaded applications

This allows you to specify that a breakpoint will only be triggered for a specific thread

Slide 41

These can be combined, note conditions always go before hit counts with filters last

Slide 42

You can also carry out actions on a breakpoint, right click and select actions, here you can display information to the console window. With these you can have the program carry on after the action or stop like a normal breakpoint. As well as display information also process code –

Slide 43

Like set the value of a variable, in the slide I have set I = 2 or call methods on a class. Note the accessibility of a method does not apply here, you can call private methods!

Slide 44

Sometimes you know a method is going to be used but you don’t know which instance is, e.g. you may have an interface implemented by several classes. You can set a breakpoint by the function name, in the breakpoints window, click on new and you can add a function breakpoint. When you start debugging you will notice that a plus sign appears next to it in the breakpoint window, which you can expand and see a breakpoint for each function. You can then disable/remove individual ones. You can add action and conditions as previously mentioned to these.

Slide 45

Not going to go into depth here, but there are windows that allow you to see memory, the disassembly (you can set breakpoints in the disassembly), and the registers. All available via debug>windows.

Slide 46

If you using WPF or UWP. There are the live visual tree and the live property explorer.

Slide 47

The visual tree allows to to look into the controls on the screen and the property tree shows the properties of the controls.

Slide 48

Dumps.

These can be opened by visual studio - just open file and select the dump.

When you open a dump, you will see the following amongst other information. I’ll just show what it looks like with managed code

Slide 49

Your see the code with the value of properties of classes, so if a program hangs - see if you can get a dump.

Slide 50

Symbol servers.

The symbol server keeps record of the pdb files giving details of symbols required for debugging. Having a symbol server allows the system to access code from for any version that has a pdb file on the server. Microsoft have 30- 40 thousand versions of windows running somewhere at Redmond and they can access the code for any one of these versions. Each build produces new pdb files, if you wish to debug a build you need the right pdb files, especially important with things that relate to production debugging like dumps.

Slide 51

There are a few tools that can help with debugging the front end. You can now debug javascript from within visual studio, But also browser link lets you run you website in a number of browsers and then change css, html and see it reflected in all browsers. This is started by using the drop down for choosing which browser, selecting browse with, select the browsers you want and clicking browse.

Browse sync - you may need to install from the marketplace. Allows when using multiple browsers to keep them in sync regarding the page and data entered.

Inspector allows you to click on a field and be taken to the line of code in visual studio. This is another marketplace add in

Slide 52

The features in the enterprise edition

Slide 53

The code map. This shows the structure of your code, you can have it built up as you debug or choose the class /method and add the references to it. You can also then click on one of the boxes and the editor will go to that segment of code, this can be very useful when debugging, for example re visit the method that was called before the one that is currently being run.

Slide 54

Intellitrace . This allows historical debugging, meaning you can go over the same scenario over and over again without having to rerun the code. First you have to turn it on and you have a choice of what you want to record. Obviously the more you want to record the larger the file, but you don’t want to miss the bit that you want to know.

Slide 55

You then have to specify the events you want recorded, again the more recorded the larger the files.

Slide 56

This is the type of output you see and you can click on the events to see where they occured.

Slide 57

You can also record a session from your live production machine, and copy the produced file to your develop machine and open it there.

And you will see something like the slide

Clicking the exception will take you to the code

Slide 58

You can jump up and down the call stack if you wish, step backwards in the code. And also do it numerous times

Slide 59

Here are some links that I got information from and that you might find useful

Slide 60

And here are some other debugging tools that I haven’t spoken about.

Though remote debugging is similar to normal debugging, just make sure no one else is using the remote server application, as they will hit your breakpoints etc. The hardest part is finding the process id out from the process you wish to attach to.

Live Debugging is IT’s version of phone a friend, and allows you to debug in you choice of tool visual studio, code on a mac or windows

The last two windbg and SOS just blow my mind.

Any questions

Breakpoints window and immediate window nse