Run zoomit

Presenton

Load visual studio focus on exceptions

Visual studio for predictor

Slide 1

Hello thank you for coming. 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

A brief history of my IT career.

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.

, stepping lines of code using the equivalent of F10 and F11 – maybe the odd run to cursor type command.

Mt twitter is there if you want My tweeting tends to be technical stuff and ice hockey.

Slide 3

These are the things I am going to go through.

Visual Studio Options

We will go through some of the settings available in Visual Studio

Attributes

Attributes that help you view information when debugging

Datatips

Provide an easy way to view information about variables

Debugging Windows

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

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 5

Debugging Options

You can see the debugging option choice either via Debug> Options, or Tools>Options>Debugging.

As you can see there are a large number of options available when you are debugging. Many enable functions and what is available. I am not going to go through them all but will try to mention various ones as we go through the talk.

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

Slide 6

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

The exception options allow you to define what happens when an exception is thrown, for visual studio to pause and break you need the appropriate exception ticked. If there is an exception that you wish to only break when it is handled. you can right click on an exception and select ‘ Continue when unhandled in user code’. A column can also be added to show this – the column is called Additional Actions.

You can search for exceptions.

Slide 7 - Demo

Demo

Conditions line 268

Two methods the same different classes – do the same thing.

Add exception search for divide add module name = Calculator2

Run code option 4 exceptions with conditions

Show Stack dump.

Go up stack dump

Slide 8

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

Slide 9

Debugger display allows you to define what is going to be show when looking at an object, property, or field is displayed in the debugger variable windows.

An example when looking at a list the default shows you the object type as seen

Slide 10

If we add a DebuggerDisplay attribute to the class, we can specify what should be shown. I have added the name property.

Slide 11

If we now look at the example the Name property is shown for each entry.

It is best to use just properties in the attribute you can use functions, but this will impact performance,

Slide 12

DebuggerBrowsable allows you to define what is shown by default for a class or a property.

Slide 13

Collapse shows you at the class level

Slide 14

Never doesn’t show the class or property at all

Slide 15

And Root Hidden, shows you the properties of the class without seeing the class level. If used with a property, then the property isn’t shown

Collapse is the default.

Slide 16

DebuggerProxy

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

Breakpoints set debugger proxy and untick the others

Line 204

Show PersonDebug the proxy for this is defined by the attribute

Then in the proxy class in the constructor take a parameter of the class and then define the property getters for the fields you want to display and how you want them displayed.

Demo option 15

Slide 17

Debugger visualisers - When you are viewing the value of a property 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.

Or you can write one yourself.

There is a template that can be used DebuggerVisualizer or you can write a class that inherits from DialogDebuggerVisualizer and override the Show method to display the item as you want.

(poss example)

Slide 18

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 19

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 20

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. Various information can be added to the display by right clicking such as line numbers, parameter types, values and names.

Slide 21

The watch window allows you to view a variable, and for classes/structures expand and see the properties and also change them as with a lot of the other variable viewing features. When passing a reference type to a function the calling code’s variable becomes out of scope, and the value is not available. If you right click the variable before the call and select make object id, it gets given an id, $1 the example in the slide. $1 then is visible from within the called function. This is useful as you can then see if the value is changed within the function. $1 will carry on showing a value until the variable is garbage collected.

Slide 22

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

Slide 23

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 24

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 25

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 26

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 27

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. There is also a task view which shows the the call stacks of System.Threading.Task.Task for managed code and also a method view that shows all threads that either call or are called by the the current method.

Slide 28

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 29

Breakpoints

The bread and butter of a lot of debugging,

There is a window for breakpoints, which shows breakpoints and whether they are active, some of the other features will be shown as we go through breakpoints.

There are several different ways to use breakpoints.

Breakpoints can be labelled which is a way of commenting them. As with datatips, breakpoints and can be exported and imported, via the breakpoint window. This is helpful when trying different scenarios or pair debugging.

Slide 30

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 wilth .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 31

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

Slide 32

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 33

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

Slide 34

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 35

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 36

Filter - for use in multi threaded applications

This allows you to specify that a breakpoint will only be triggered for a specific thread – options MachineName, ProcessId, ProcessName, ThreadId, and ThreadName. For this to work you need Enable breakpoint filters set in the debugging options

Slide 37

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

Slide 38

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 39

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 40

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 41

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.

For these options to be available you need the **Enable address-level debugging**: option selected.

Slide 42

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

Slide 43

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

Slide 44

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 45

Your see the code with the value of properties of classes, so a dump can be very useful.

Ctrl ET - resharper

Slide 46

Symbol servers.

In cases such as dumps, or components where the code isn’t in your project how does visual studio know how to show the correct code. It may well be that it is easy and you only have 1 version of code, but what if some clients are on older versions. In 2014 Microsoft had 30-40,000 different versions of windows running at Redmond, I assume that figure is similar now, but they could access the code for each build, with the help of Symbol Servers.

The symbol server keeps a record of the pdb files giving details of symbols required for debugging. Each build produces new pdb files, these aren’t only produced in debug mode but also in release mode. if you wish to debug a build you need the right pdb files.

When you build your solution, you can publish your symbols, in Azure Devops there is a task for this, you can either publish to your own symbol server, or use Azure Devops. Then in Visual studio under Debug>Options there is a symbol server option where you can enter the location of your symbol server. There is also a tick box in the debug options, source server support, if you don’t tick this and the server is different to the machine you are running visual studio on it looks for the location on your machine. Then when you try to step into your code the source will be found.

Not only do they help with debugging your own code, they allow you to step into 3rd party code. If you know the symbol server for the 3rd party add that into visual studio and away you go.

Using the Microsoft symbol server you can debug through the .net framework (you will need enable .net framework source stepping in the options.

You will also need to untick Just My code, Visual studio uses dbg, pdb and optimisation to determine if code is your code.

With nuget packages there is now soucelink which builds into the pdb metadata to remap to files on github, gitlab or AzureDevops. You will need SourceLink ticked in the debug options.

Slide 47

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 48

The features in the enterprise edition

Slide 49

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 50

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 51

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

Slide 52

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

Slide 53

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 54

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

Slide 55

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

Slide 56

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. Work processes (Mike Warren)

Immediate and attributes

Live Debugging is IT’s version of phone a friend, and allows you to debug in your 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