Run zoomit

Switch off all exceptions.

Clear Breakpoints

Check format exception no continue user handled code

Debug options enable just my code.

Raw switch off

Properties switch on

Account Holder clear from debugging attributes

Presentation

Slide 1 – 40 seconds

Hello thank you for coming. I’m Philip and I’m going to go through some of the debugging features available in Visual Studio.

Whether it is greenfield development or not, debugging is something we have to do. It 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 1-40

A bit about me

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

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.

Slide 3 2-40

Before I start the code is often written to give examples of debugging techniques and problems – not necessarily the way I would write code

Whether it is greenfield development or not, debugging is something we have to do. It 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.

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

Breakpoints

Go through the various types of breakpoints

Debugging Windows

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

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

Examining dumps – how to use dumps with visual studio

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.

Debugging Options 3mins + 6mins – 9 mins

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

As you can see there are many 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

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

The exception options allow you to define what happens when an exception is thrown. So what does happen in visual studio, if the exception is not handled visual studio stops at the line that throws the exception. If the exception is handled, visual studio doesn’t stop. For visual studio to pause and break you need the appropriate exception ticked.

If for example in web applications they handle error messages themselves. When running the code will stop if the error is unhandled – no try catch. However they is an option in the settings that allows you to say that if the code is unhandled in user code it will continue – this is only available when just my code is ticked – one of the debugging options.

You can search for exceptions by entering in the search box.

You can also specify via conditions that then break should only occur in set modules.

Code Demo

Demo

We have 2 classes that implement the InterestCalculator interface and both have the divide by zero – this gives the exception Overflow with doubles – but you also need to have the flag ticked in the properties for the project in the advanced section, otherwise you get an answer in this case that is very wrong.

Condition to stop in Module HighInterestCalculator.dll

Show with B

Show with H.

Slide 5 Debugging Attributes m19 mins

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

[DebuggerDisplay("Name = {Name}")]

…

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, If we use the search facility as well you will see the debugger display value is searched.

DEBUGGER BROWSABLE -

Allows you to specify how when you are looking at classes and properties in variable windows – how they appear.

Collapsed is the default – so if you have a nested class when viewing through a variable window – only the top level of the nested class is shown.

With RootHidden the properties of the nest class are shown but not the class, and then Never nothing about the nested class is shown At the property level RootHidden and Never mean you don’t see the property.

Debugger Proxy

Allows the significant and fundament change on the debugging view of a type.

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

Show code for AccountHolderProxy

Quite easy to write

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.

Add [DebuggerTypeProxy(typeof(AccountHolderProxy))]

The raw view ignores this -

Slide 7 Breakpoints 34min

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.

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.

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. In core 3.0 there is an option to set a breakpoint on a variable e.g. through the watch window right click on the variable and set break on value changes. The menu option is there but only available in 3. This is different to setting a breakpoint on a set. On a set this with stop whenever that property is set – irrelevant if the value is actually changing, and also this is on the property not an instance of a variable. You could add a condition to the breakpoint so check the id of the class or if the value as changed

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.

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.

Filter

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

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

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 –

Like set the value of a variable, Note the accessibility of a method does not apply here, you can call private methods!

Break at Function

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.

For example I have the interest calculator – I can break when the method to calculate the interest is hit, In the breakpoint window I can select Break At Function. Conditions etc ca be set for each method

Option 2 as example

Slide 8 40 mins

The watch auto and local windows allow you to look at the value of variables. With the watch window you add variables of your choice, the local window focuses in on the variables used in a method, and the autos narrows down to variables that are used in the current line and the 1 or two before that. One of the lesser known features is the MakeObjectId. This is helpful when a reference type variable is being passed about. When a reference variable is passed into a method your up to date view of it disappears. In the watch window it remains there but in 2019 what you see are the values of the variable at the time of the call. In 2017 I think it said out of scope. So if the value of it changes in the method you might not notice. However DEMO if you right click the variable and makeobject id you will see along side the variable A $, if locals and autos this is added in the watch window you need to add it. Now when the value is changed in a method it is reflect in the $1.

There is now better support for linq in locals and autos.

New in VS2019 is the facility to search. A particular use of the search is with collections when by default all you know is the the type when looking. The search facility allows you to find the entry that refers to a particular instance e.g. if you have a collection of Person you can enter a persons name in and that will be shown.

The depth of search can be change – the deeper you look the longer the search can take.

Call Stack

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. Beakpoints can be set here this will create a breakpoint on the immediate return to this level – before any other processing has happened on that line.

Slide 9 47

Multithreading 9

Debugging multithreaded application can be awkward. There are a number of windows that can help, I will go through them and hopefully give some indication of when it is best to use each of the windows.

Debug ToolBar

Not a window but a toolbar. Most of it I’d expect you to know – the start, restart, step over , step into and step out of button. How ever on the end is a button that shows you where each of your threads are currently. It works in VS2017 but in VS2019 from 16.2 For threads you can mark thread as flagged. These threads are the ones you are interested in.

DebugLocation

Another toolbar this on the debug location. You may need to add this to see it.

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. You can see which thread is running at the moment by the >. You can limit the display to those threads you have flagged

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 threads using the search window, show different details by selecting what columns to show, and freeze and thaw threads. uou can also flag columns and focus in on those.

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. Do however b e careful when freezing threads as this may change the way the program runs compared to live e.g. you won’t get a deadlock if you only have 1 thread running.

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.

Tasks deadlock

When you are using tasks the task window shows the tasks and also if they are blocked or deadlocked. DEMO

Stacks

If you look a task and double click it with the stack window showing you will see the stack window showing the call stack for the thread focused.

With the parallel stacks view if you select threads you seen window for each thread and the call stack in one view. Threads can be frozen and thawed here as well

The task view shows the call stack if using tasks. There is the thread symbol to show you which is active at the moment.

Parallel watch windows allows you to see the values of variable that there are multiple instances of when you are running in parallel.

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 10

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. Microsoft can access the code to every built version of windows which is in the thousands, 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 11

Dumps, a dump from a .net program can be loaded into visual studio, the line when the dump is taken is shown and some values of variables can be seen. Symbols are required for this.

Slide 12 summary

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