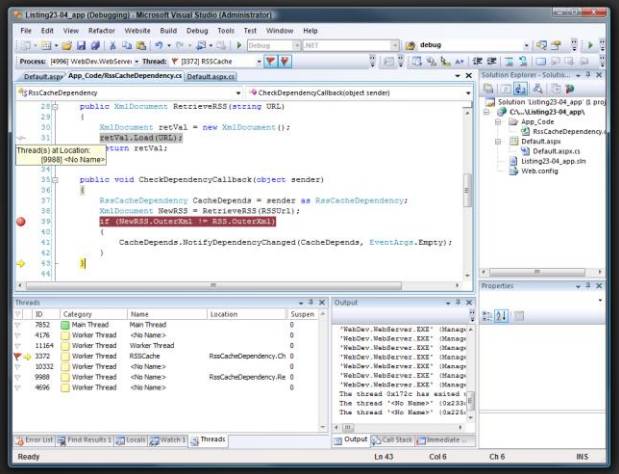
Telerik Academy



Debugging

Building Rock-Solid Software

Telerik Software Academy

Learning & Development

http://academy.telerik.com

Telerik Academy

Table of Contents

- Introduction to Debugging
- Visual Studio Debugger
- Breakpoints
- Data Inspection
- Finding a Defect

Telerik Academy

Introduction to Debugging

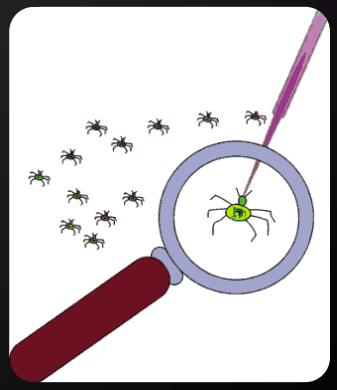
Telerik Academy

What is Debugging?

- The process of locating and fixing or bypassing bugs (errors) in computer program code
- To *debug* a program:
  - start with a problem
  - isolate the source of the problem
  - fix it
- Debugging tools (called *debuggers*) help identify coding errors at various development stages

Debugging vs. Testing

- Testing
  - A means of initial detection of errors
- Debugging
  - A means of diagnosing and correcting the root causes of errors that have already been detected



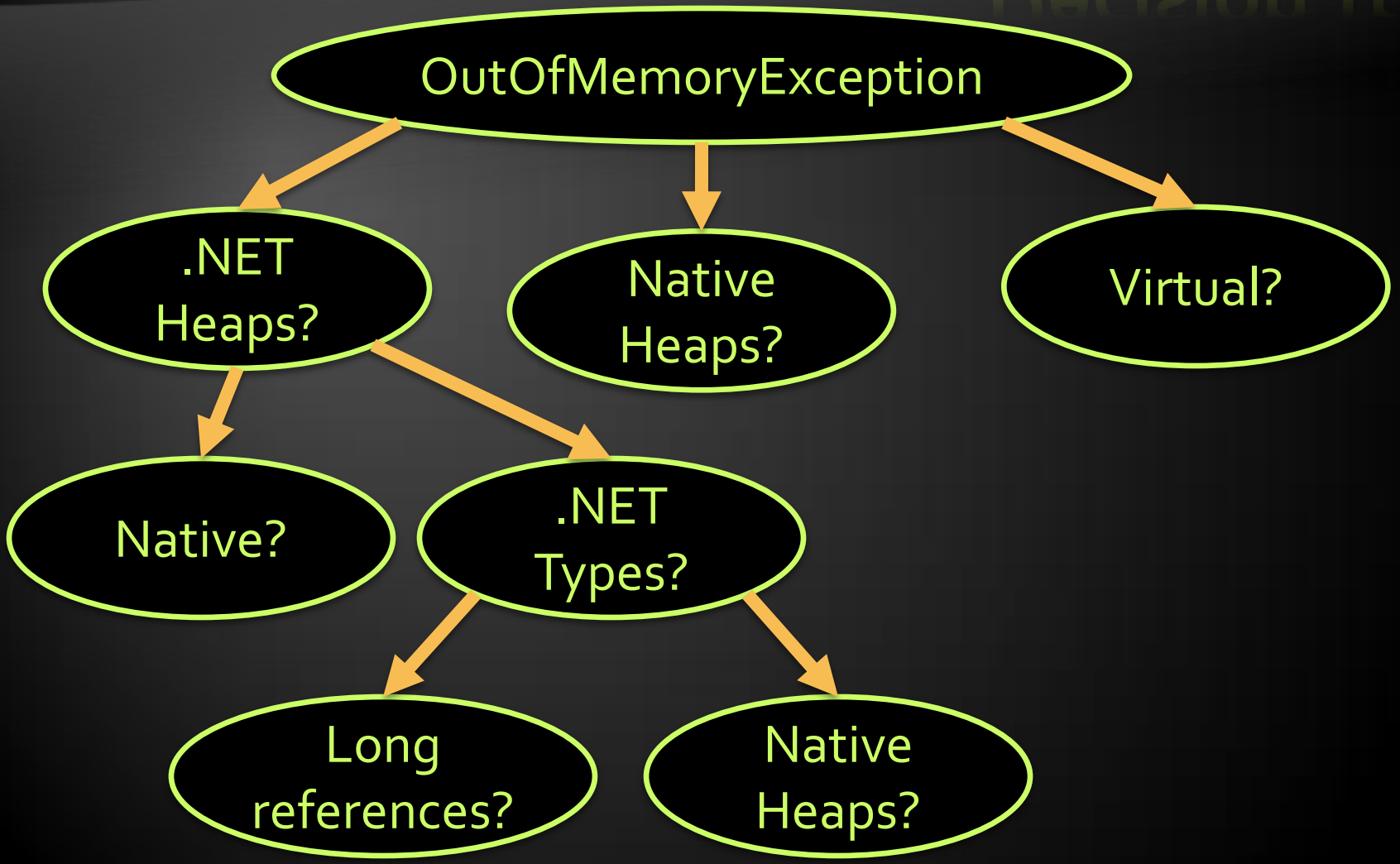
Importance of Debugging

- $60 Billion per year in economic losses due to software defects
- Perfect code is an illusion
  - There are factors that are out of our control
- Legacy code
  - You should be able to debug code that is written years ago
- Deeper understanding of system as a whole

Debugging Philosophy

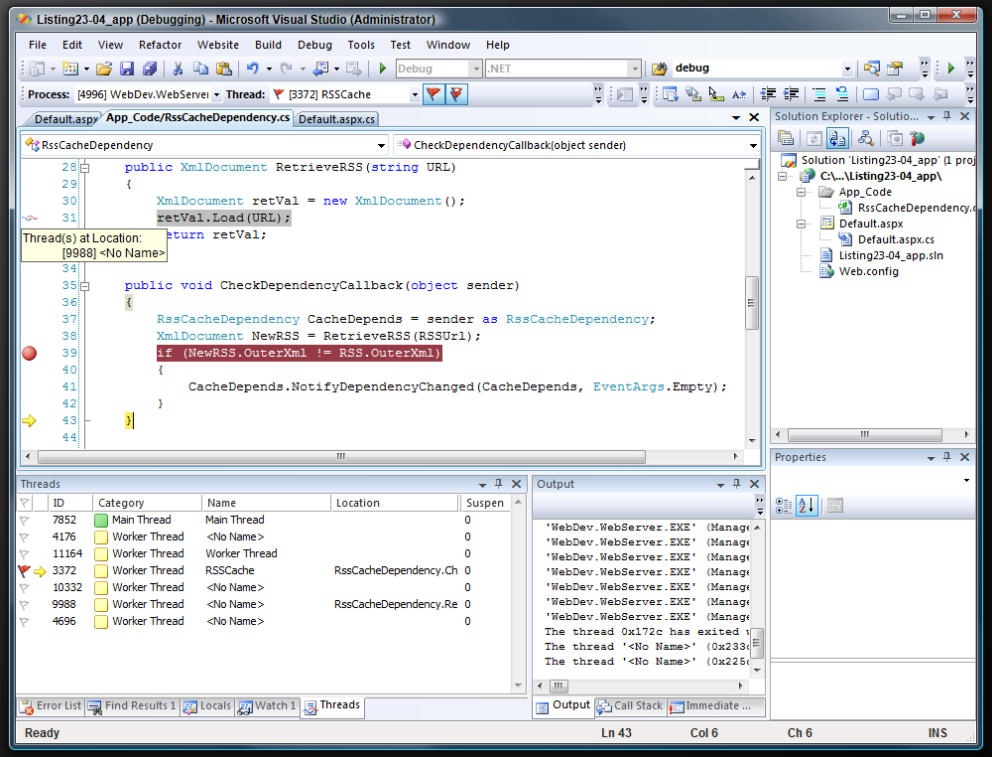
- Debugging can viewed as one big decision tree
  - Individual nodes represent theories
  - Leaf nodes represent possible root causes
  - Traversal of tree boils down to process state inspection
  - Minimizing time to resolution is key
    - Careful traversal of the decision tree
    - Pattern recognition
    - Visualization and easy of use helps minimize time to resolution

Example Debugging Decision Tree



Telerik Academy

Visual Studio Debugger



Telerik Academy

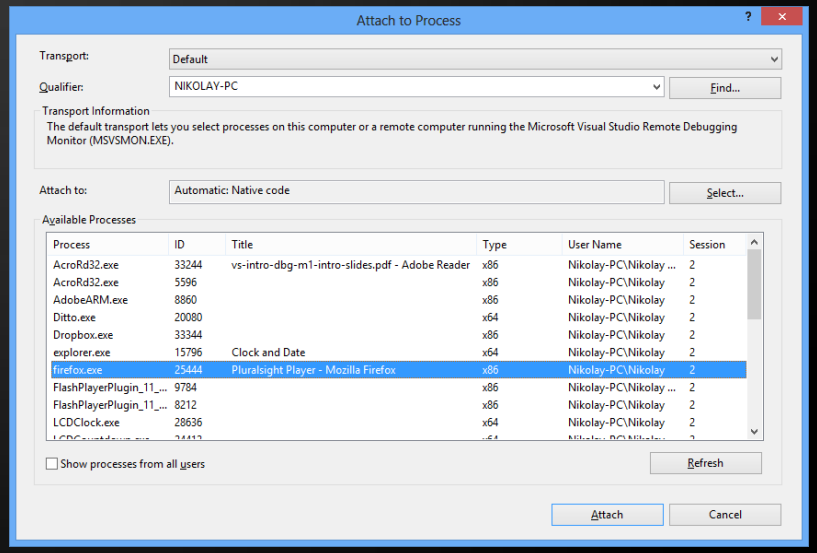
Visual Studio Debugger

- Visual Studio IDE gives us a lot of tools to debug your application
  - Adding breakpoints
  - Visualize the program flow
  - Control the flow of execution
  - Data tips
  - Watch variables
  - Debugging multithreaded programs
  - and many more…

Telerik Academy

How To Debug a Process

- Starting a process under the Visual Studio debugger
- Attaching to an already running process
  - Without a solution loaded you can still debug
  - Useful when solution isn't readily available
  - Debug menu -> Attach to Process



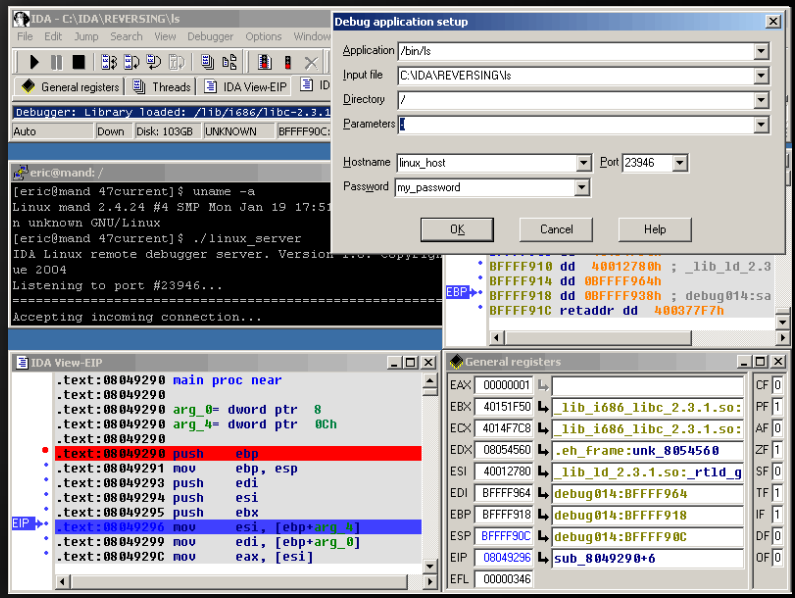
Debugging a Solution

- Debug menu, Start Debugging item
  - F5 is a shortcut
- Easier access to the source code and symbols since its loaded in the solution
- Certain differences exist in comparison to debugging an already running process
  - Hosting for ASP.NET application
    - VS uses a replacement of the real IIS

Telerik Academy

Debug Windows

- Debug Windows are the means to introspect on the state of a process
- Opens a new window with the selected information in it
- Window categories
  - Data inspection
  - Threading
- Accessible from menu
  - Debug -> Windows



Debugging Toolbar

- Convenient shortcut to common debugging tasks
  - Step into
  - Step over
  - Continue
  - Break
  - Breakpoints

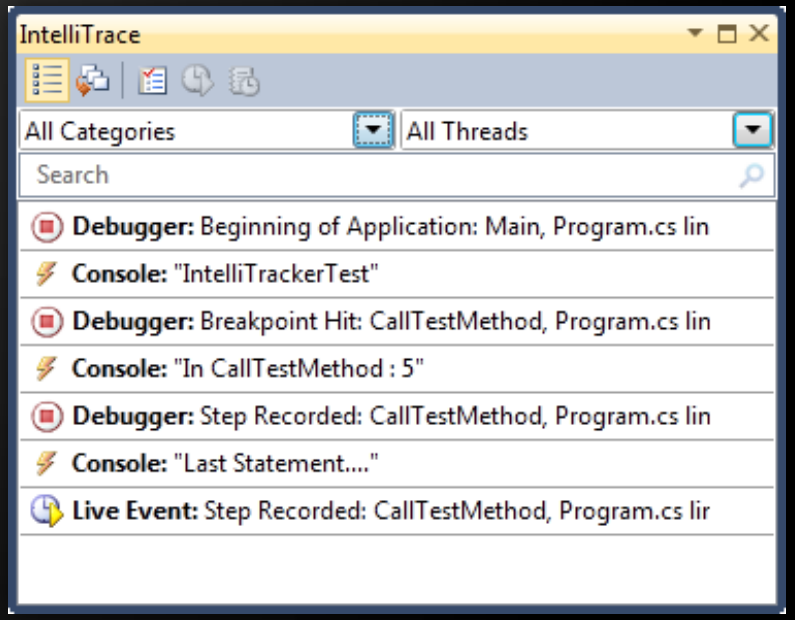
- Customizable to fit your needs
  - Add and/or remove buttons

Controlling Execution

- By default, an app will run uninterrupted (and stop on exception or breakpoint)
- Debugging is all about looking at the state of the process
- Controlling execution allows:
  - Pausing execution
  - Resuming execution
  - Stepping through the application in smaller chunks
  - In the case of IntelliTrace (recording steps), allows backward and forward stepping

IntelliTrace

- IntelliTrace operates in the background, records what you are doing during debugging
- You can easily get a past state of your application from intelliTrace
- You can navigate your code with any part and see what's happened
  - To navigate, just click any of the events that you want to explore



Options and Settings

- Visual Studio offers quite a few knobs and tweaks in the debugging experience
- Options and settings is available via Debug -> Options and Settings
- Examples of Options and Settings
  - Enable just my code (ignore other code)
  - Enable .NET framework source stepping
  - Source server support
  - Symbols (line numbers, variable names)
  - Much more…

Telerik Academy

Breakpoints

Breakpoints

- Ability to stop execution based on certain criteria is key when debugging
  - When a function is hit
  - When data changes
  - When a specific thread hits a function
  - much more
- Visual Studio debugger has a huge feature set when it comes to breakpoints

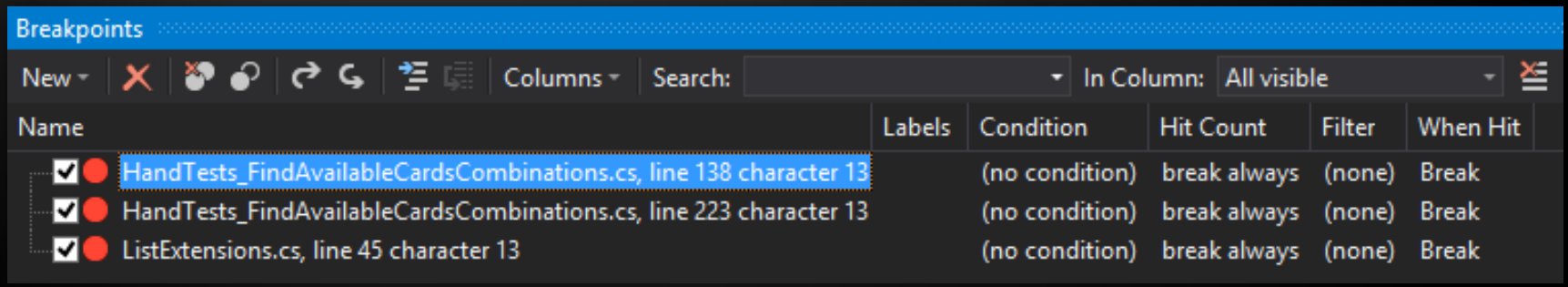
Visual Studio Breakpoints

- Stops execution at a specific instruction (line of code)
  - Can be set using Debug->Toggle breakpoint
    - F9 shortcut
    - Clicking on the left most side of the source code window
- By default, the breakpoint will hit every time execution reaches the line of the code
- Additional capabilities: condition, hit count, value changed, when hit, filters

Telerik Academy

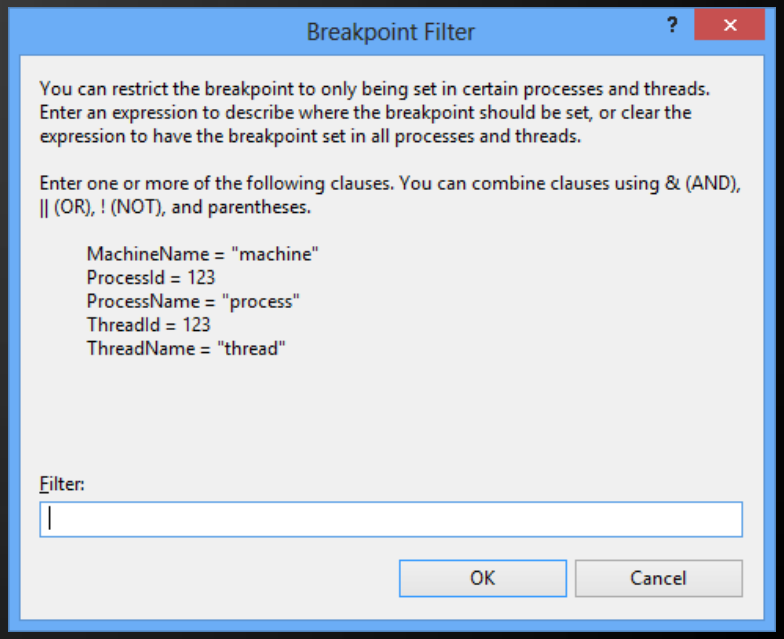
Managing Breakpoints

- Managed in the breakpoint window
- Adding breakpoints
- Removing or disabling breakpoints
- Labeling or grouping breakpoints
- Export/import breakpoints



Breakpoint Filters

- Allows you to excerpt even more control of when a breakpoint hits
- Examples of customization
  - Machine name
  - Process ID
  - Process name
  - Thread ID
  - Thread name



- Multiple can be combined using &, ||, !

Telerik Academy

Data Inspection

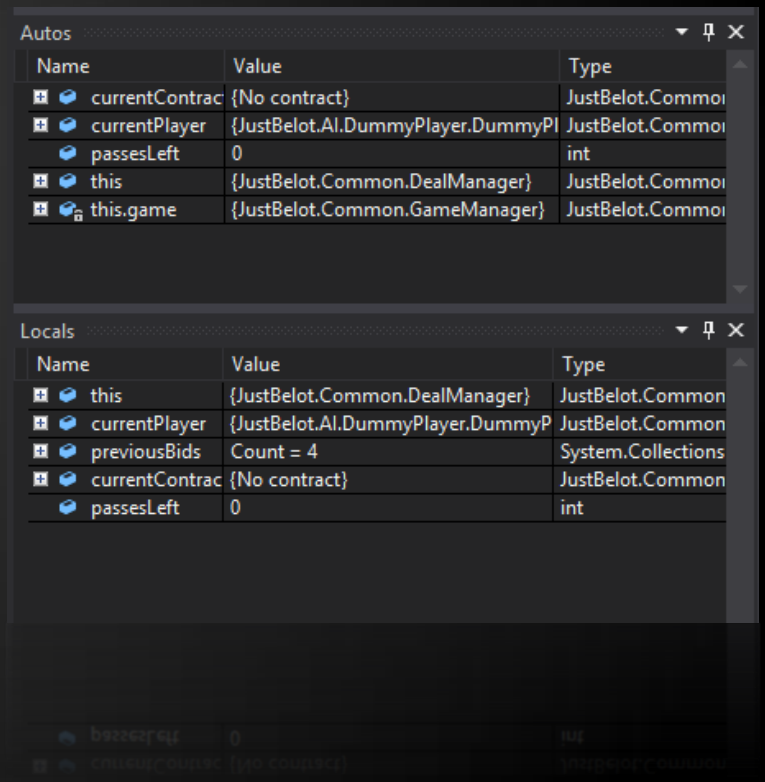
Data Inspection

- Debugging is all about data inspection
  - What are the local variables?
  - What is in memory?
  - What is the code flow?
  - In general - What is the state of the process right now and how did it get there?
- As such, the ease of data inspection is key to quick resolution of problems

Telerik Academy

Visual Studio Data Inspection

- Visual Studio offers great data inspection features
  - Watch windows
  - Autos and Locals
  - Memory and Registers
  - Data Tips
  - Immediate window



Watch Window

- Allows you to inspect various states of your application
- Several different kinds of "predefined" watch windows
  - Autos
  - Locals
- "Custom" watch windows also possible
  - Contains only variables that you choose to add
  - Right click on the variable and select "Add to Watch"

Autos and Locals

- Locals watch window contains the local variables for the specific stack frame
  - Debug -> Windows -> Locals
  - Displays: name of the variable, value and type
  - Allows drill down into objects by clicking on the + sign in the tree control
- Autos lets the debugger decide which variables to show in the window
  - Loosely based on the current and previous statement

Memory and Registers

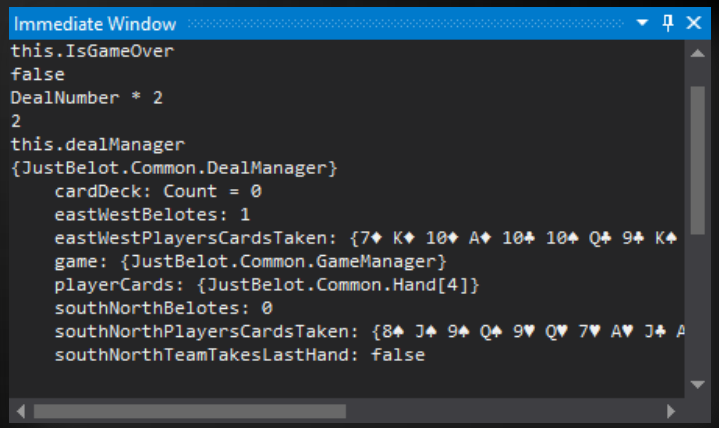
- Memory window can be used to inspect process wide memory
  - Address field can be a raw pointer or an expression
  - Drag and drop a variable from the source window
  - Number of columns displayed can be configured
  - Data format can be configured
- Registers window can be used to inspect processor registers

Data Tips

- Provides information about variables
  - Variables must be within scope of current execution
- Place mouse pointer over any variable
  - Variables can be expanded by using the + sign
- Pinning the data tip causes it to always stay open
- Comments can be added to data tips
- Data tips support drag and drop
- Importing and exporting data tips

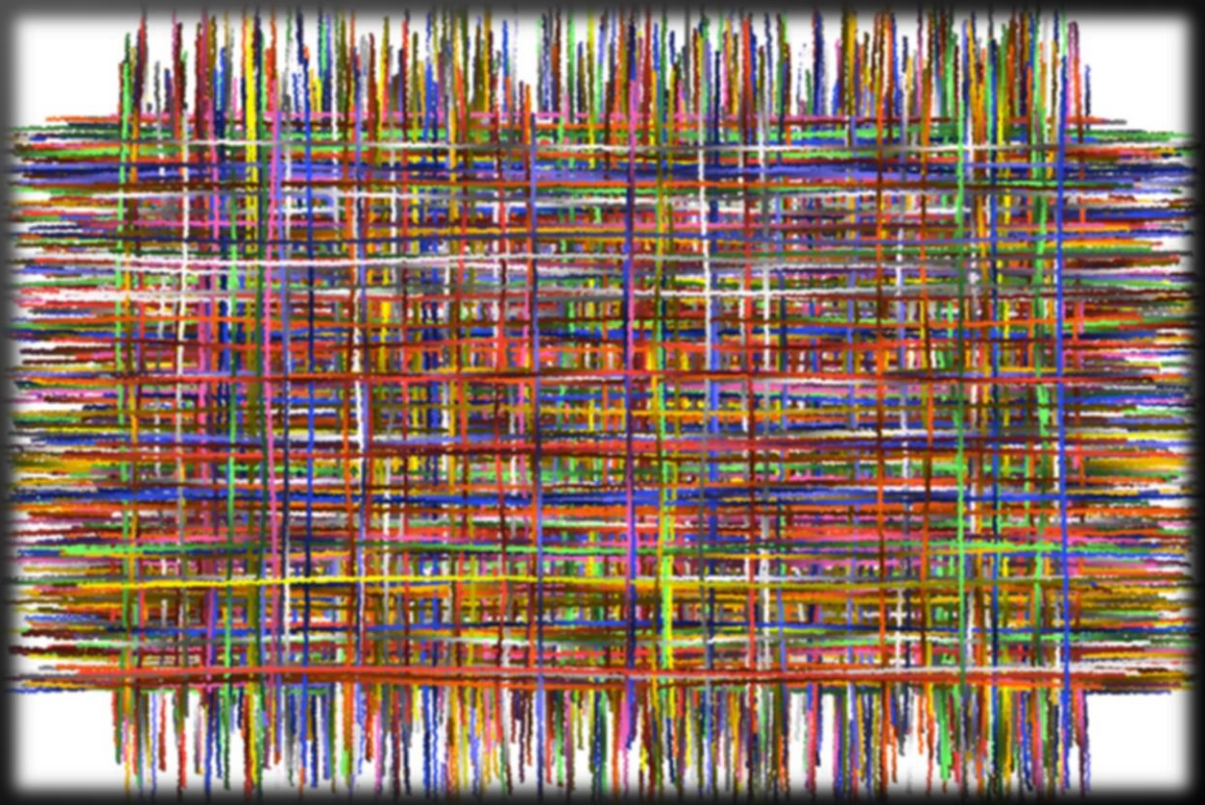
Immediate Window

- Useful when debugging due to the expansive expressions that can be executed
  - To output the value of a variable <name of variable>
  - To set values, use <name of variable>=<value>
  - To call a method, use <name of variable>.<method>(arguments)
  - Similar to regular code
  - Supports Intellisense



Telerik Academy

Threads and Stacks

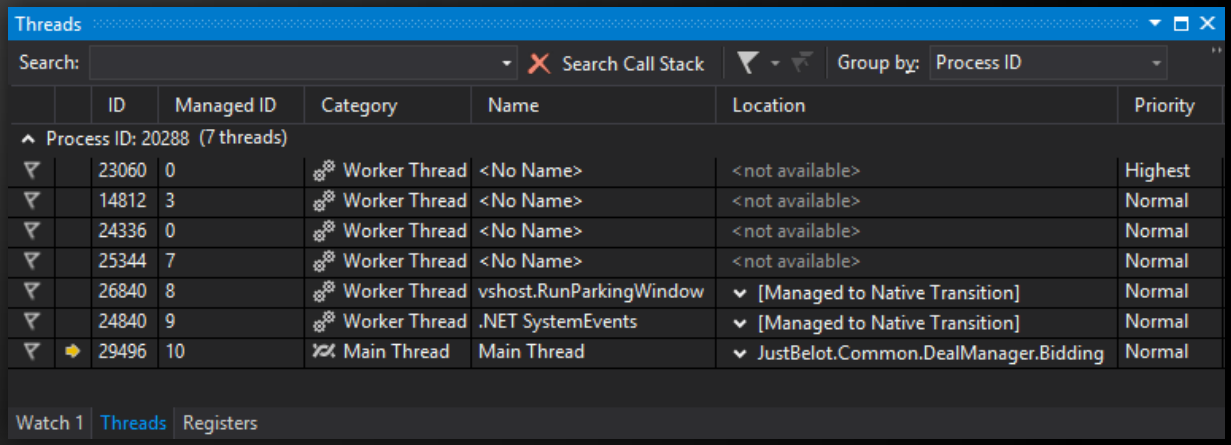


Threads

- Fundamental unit of code execution
- Commonly, more than one thread
  - .NET, always more than one thread
- Each thread has a memory area associated with it known as a stack used to
  - Store local variables
  - Store frame specific information
- Memory area employs last in first out semantics

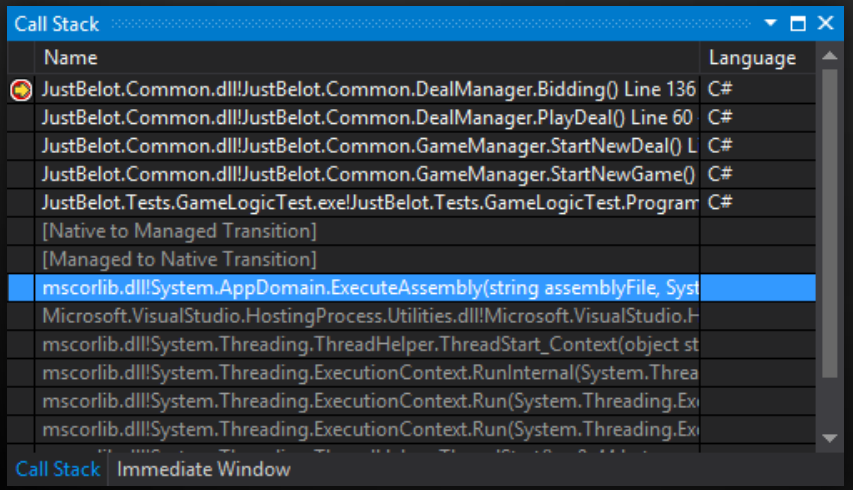
Threads Window

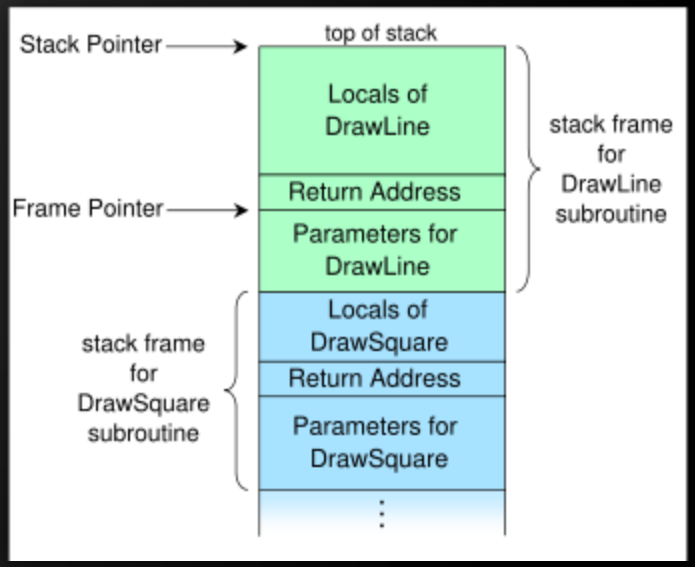
- Contains an overview of thread activity in the process
- Includes basic information in a per thread basis
  - Thread ID's
  - Category
  - Name
  - Location
  - Priority



Callstacks

- A threads stack is commonly referred to as a callstack
- Visual Studio shows the elements of a callstack
  - Local variables
  - Method frames





Telerik Academy

Finding a Defect

Finding a Defect

1. Stabilize the error
2. Locate the source of the error
   a) Gather the data
   b) Analyze the data and form hypothesis
   c) Determine how to prove or disprove the hypothesis
   d) Prove or disprove the hypothesis by 2c
3. Fix the defect
4. Test the fix
5. Look for similar errors

Tips for Finding Defects

- Use all available data
- Refine the test cases
- Check unit tests
- Use available tools
- Reproduce the error several different ways
- Generate more data to generate more hypotheses
- Use the results of negative tests
- Brainstorm for possible hypotheses

Tips for Finding Defects (2)

- Narrow the suspicious region of the code
- Be suspicious of classes and routines that have had defects before
- Check code that's changed recently
- Expand the suspicious region of the code
- Integrate incrementally
- Check for common defects
- Talk to someone else about the problem
- Take a break from the problem

Fixing a Defect

- Understand the problem before you fix it
- Understand the program, not just the problem
- Confirm the defect diagnosis
- Relax
- Save the original source code
- Fix the problem not the symptom
- Make one change at a time
- Add a unit test that expose the defect
- Look for similar defects

Source: http://www.movingseniorsbc.com

Psychological Considerations

- Your ego tells you that your code is good and doesn't have a defect even when you've seen that it has one.
- How "Psychological Set" Contributes to Debugging Blindness

Telerik Academy

Debugging

Questions?

http://academy.telerik.com

Telerik Academy

Free Trainings @ Telerik Academy

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com