# The Essential Toolbox for Troubleshooting ASP.NET Web Applications

Ido Flatow
Senior Architect, Microsoft RD & MVP



IT/Dev CONNECTIONS

# How many of you have ever had …

# A Crash?

# High memory usage or OOM?

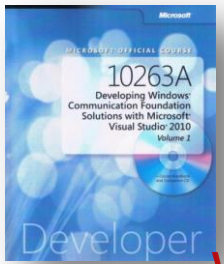
# Slow response times in your application?

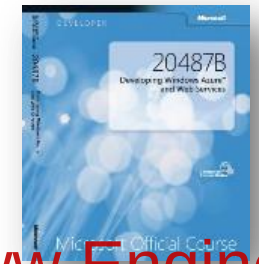
# Unexplained exceptions?

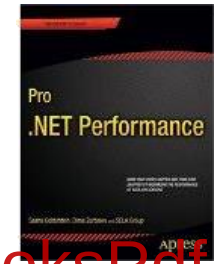
# Issues that only occur in production?

# About Me

- Senior Architect, Sela Group
- Microsoft Regional Director, and MVP
- Co-author of courses and books
- Focus on server, web, and cloud




# By Day's End

- Tools of the trade for
  - Logs
  - Code decompilers
  - Performance counters
  - Dumps
  - Profilers
  - Network sniffers
  - Sysinternals tools
- Which tools to use for
  - Unwanted exceptions
  - Crashes
  - Slow response time / Hangs
  - High memory usage

You'll mostly be exhausted

# SERVER-SIDE EXCEPTIONS

# Troubleshooting Tools

- Logs
  - IIS log files and Failed Request Tracing
  - ASP.NET messages in Event Viewer
  - Application logs
- Network sniffers
  - Fiddler
  - Browser DevTools
- De-compilers
  - ILSpy / JustDecompile / Reflector
- Dump on exceptions
  - DebugDiag

Being able to reproduce an exception brings you one step closer to resolving it

# Historical Logs

# Understanding IIS Logs

- IIS has the ability to write client and server activity information to a text-based log file
- IIS does not include tools that:
  - Query the log files
  - Aggregate log file data
- The default format IIS uses is called the extended W3C format

# Configuring W3C Logging

- Logging can be configured:
  - At the server level, and that configuration will affect all websites
  - At the website level, and that configuration overrides the server configuration
- By default, log files are stored in %Systemdrive%\inetpub\logs\LogFiles
- IIS creates a subfolder for each website
- When using W3C logging, you can configure IIS to include more or fewer pieces of log information

# Advanced Logging

- IIS logging is not enough!
- Use the Advanced Logging module
  - http://bit.ly/iis-advanced-logging
- Create a separate log file per application
- Log HTTP headers, performance counters, and server variables
- Filter requests you don't want to log
- Creates the same IIS Log file structure (almost)
  - Use your existing analysis tools

# What can we Learn from Logs?

- Look for HTTP 4xx-5xx responses
- Check the sub-status code
  - http://support.microsoft.com/en-us/kb/943891
- Is it happening each time for the same URL? Specific query string?
- Is it limited to a specific timeframe?
- Correlate this data with the application log
- Remember - record time is in GMT

# Installing Log Parser

- IIS does not include any tools that make it easier to retrieve information from the IIS log files
- Microsoft offers a free tool called Log Parser that can be used to scan log files and produce query results
- Log Parser is a command-line tool
  - http://www.microsoft.com/en-us/download/details.aspx?id=24659
- Log Parser Studio is a graphical tool that adds an easier to use graphical user interface (GUI) over Log Parser
  - https://gallery.technet.microsoft.com/office/Log-Parser-Studio-cd458765

# Failed Request Tracing (FREB)

- Investigate failed/bad requests
- What is a bad request?
  - Resulted in a 4xx or 5xx response
  - Took too long to process
  - Caused a warning or error trace message
- You can set different traces for different URLs
- Buffers all trace output, but only flushes to disk if request fails
- Use the **System.Web.IisTraceListener** trace listener for custom traces
- Beware of overuse!
- View traces easily with the Trace Viewer extension
  - http://www.iis.net/downloads/community/2008/03/iis-70-trace-viewer

# HTTP.SYS Log Files, Wait, What's HTTP.SYS?

- It's the thing that listens to HTTP on your computer
- It's a kernel mode device driver
- Ever since Windows Server 2003 (IIS 6)
- Responsible for:
  - Routing requests to registered applications
  - Kernel-mode SSL (as of Windows Server 2008)
  - Response caching in kernel mode
  - Request queue per application pool
  - QoS, such as connection limits and timeouts
- Want to know more? **netsh http show**

# HTTP.SYS Log Files

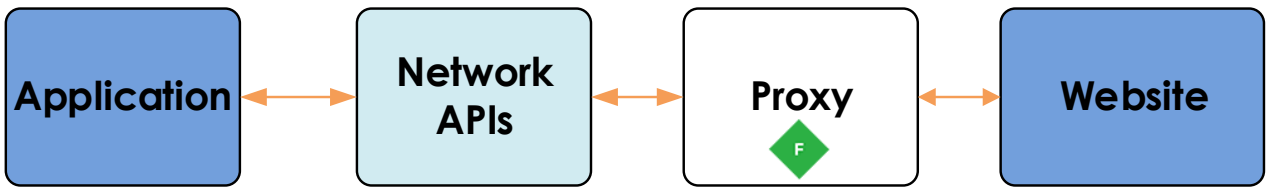
- Located in Windows\System32\LogFiles\HTTPERR
- 1MB per file
- What does it log? Mainly faulty connections/requests
  - Connection_Abandoned_By_AppPool – AppPool crashed
  - Connection_Dropped – Closed by client prior to response
  - Timer_* (ConnectionIdle, HeaderWait, AppPool,…) – Timeouts
  - Various parse errors (request length, incorrect verb, …)
  - HTTP 500
  - HTTP 503 (N/A, ConnLimit, QueueFull, Disabled, …)

# Application-Specific Diagnostics

- ASP.NET page traces integrate into FREB logging
- System.Diagnostics.Trace
  - Capture tracing information from .NET components
  - Configure trace sources to control specific application tracing information
- ASP.NET Health Monitoring events in Event Viewer
- Your own custom logs (Log4Net, Logging Application Block, ELMAH)
- Correlate and analyze application tracing information in the context of the overall request

# Sniffing Tools

# What is Fiddler?



# Fiddler to the Rescue

- Browsers (Static sites, ASP.NET, J2EE, PHP)
- Desktop applications that use HTTP
- Web services
- Smartphone emulators
- Any device that supports a proxy server ( )
- Download from: http://www.fiddlertool.com

# Learning from Fiddler

- See content of requests and responses (header & body)
- Inspectors transform content to viewable form
- Can listen to both HTTP and HTTPS
- Advanced filtering and search tools
- Replay requests for easy debugging
- Composing Ad-Hoc requests
- Store entire request/response list for later inspection

# If you Cannot Fiddle, press F12

# Other Tools for Sniffing

- Network capture tools
  - Wireshark
    https://www.wireshark.org
  - Microsoft Message Analyzer (previously Network Monitor)
    http://www.microsoft.com/download/details.aspx?id=44226
- HTTP-specific
  - HTTP Watch
    http://www.httpwatch.com
  - Firebug
    http://getfirebug.com
  - Charles Proxy (Win/Mac/Linux)
    http://www.charlesproxy.com

# Pro-Active with Dumps

# Dump Files

- A user dump is a snapshot of a running process
- Dump files are useful for post-mortem diagnostics and for production debugging
- Anytime you can't attach and start live debugging, a dump might help

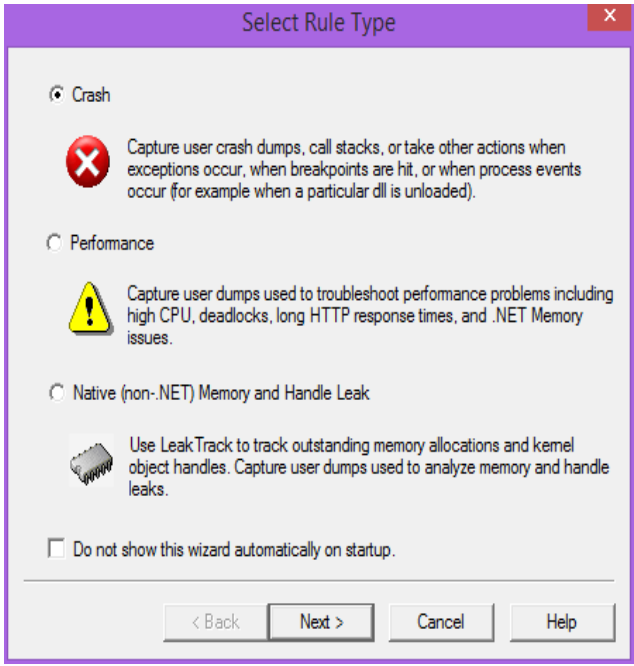
You can save a dump, move it around, and analyze it later.
You can't "debug" it

# Dump on Exception

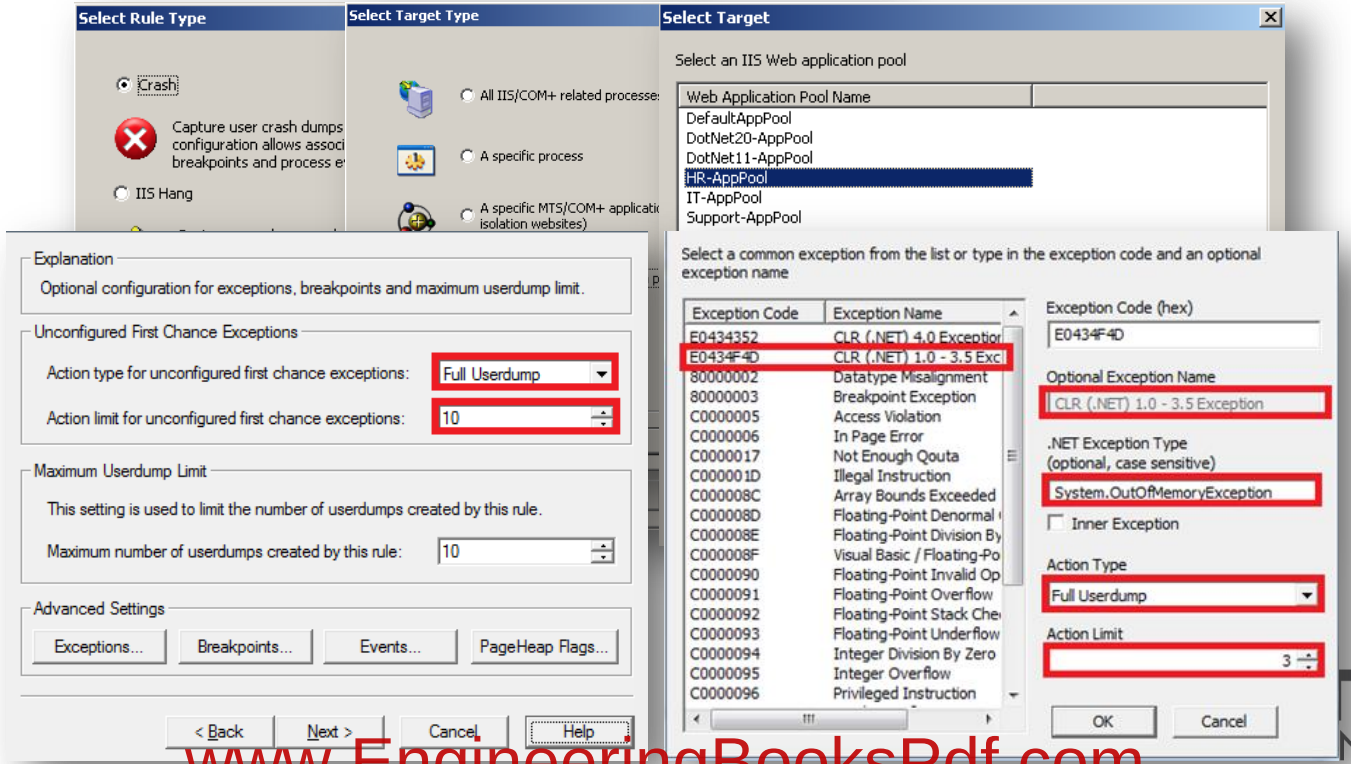
- Use a tool (DebugDiag) to collect a dump on first chance exception
- Configure when/what to collect
  - Specific exception type
  - Dump type (mini/full)
- Disable the IIS Application Pool ping
- Wait for it…
- Open in a dump analyzer and examine
  - Exception message
  - Call stack
  - Objects value

# DebugDiag

- Microsoft tool for monitoring and dump generation
  - Very suitable for ASP.NET
  - Dump analysis component included
- Download from: http://www.microsoft.com/en-us/download/details.aspx?id=42933

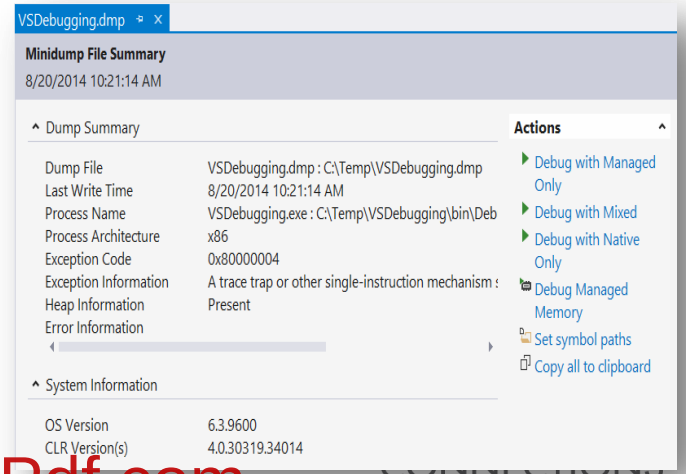


# Configuring DebugDiag



# Opening Dump Files

- Visual Studio can open dump files
- For .NET, CLR 4.0+ and VS2010+ required
- Managed memory debugging requires Enterprise edition



# Taxonomy of Dumps

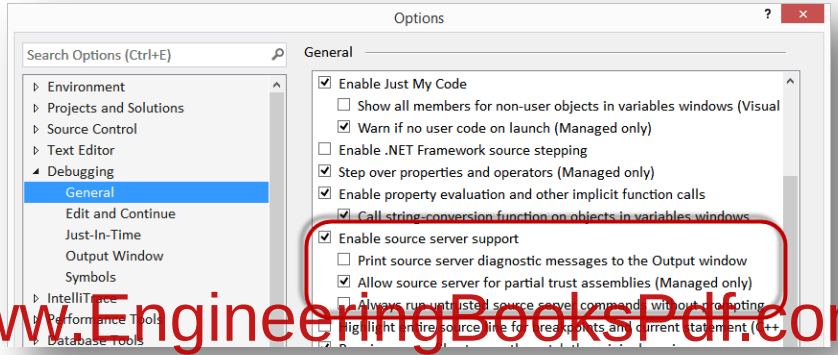
- Crash dumps are dumps generated when an application crashes
- Hang dumps are dumps generated on-demand at a specific moment
- These are just names; the contents of the dump files are the same!

# Debugging Symbols

- Debugging symbols contain source file names and line numbers
  - PDB files
  - Required for proper debugging and dump analysis
- Visual Studio is already configured to use Microsoft's Symbol server
- For many 3rd-party components and new ASP.NET libraries, use the SymbolSource symbol server:
  - http://srv.symbolsource.org/pdb/Public
- If your machine is not connected to the Internet, you will need to build your own Symbol Server
  - https://msdn.microsoft.com/en-us/magazine/cc301459.aspx

# Viewing .NET Code in Dumps

- Viewing code in Visual Studio while analyzing a dump is useful
- To view your own code, make sure it exists at the location specified in the .pdb file
- To view the .NET Framework and ASP.NET source code, enable source server

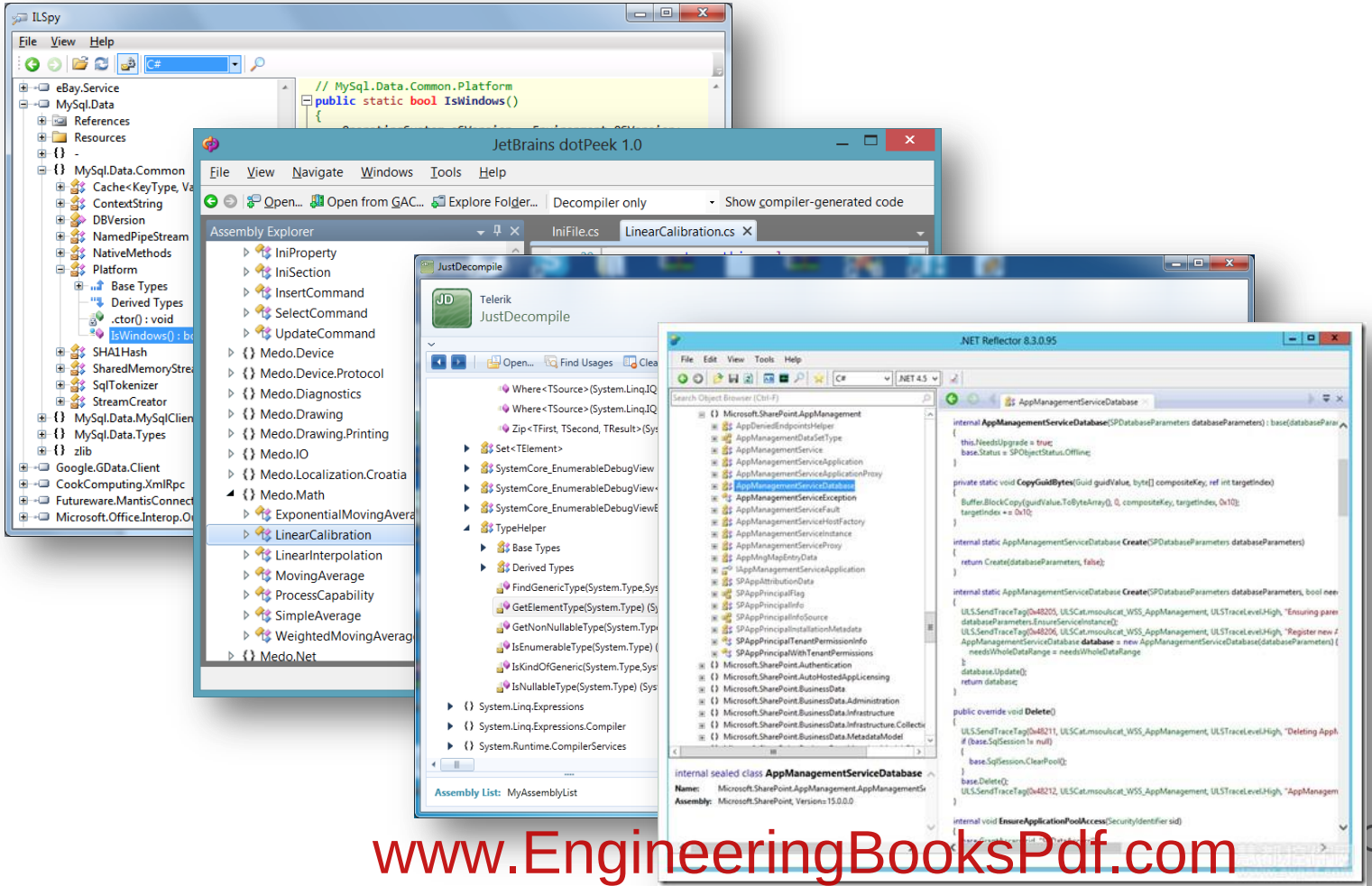


# Code Decompilation

# What are Code Decompilers?

- Reconstructs the source code (to some extent) from the compiled code
- .NET assembly → C# source code
- Reconstructed code is not always 100% identical to source
- Inspecting code "offline", without a debugger attached
- Way better than reading code in GitHub
- Many tools can do the job
  - ILSpy (open source)
  - dotPeek (JetBrains, free)
  - JustDecompile (Telerik, free)
  - Reflector (Redgate, not free)

# THE ESSENTIAL TOOLBOX FOR TROUBLESHOOTING ASP.NET WEB APPLICATIONS

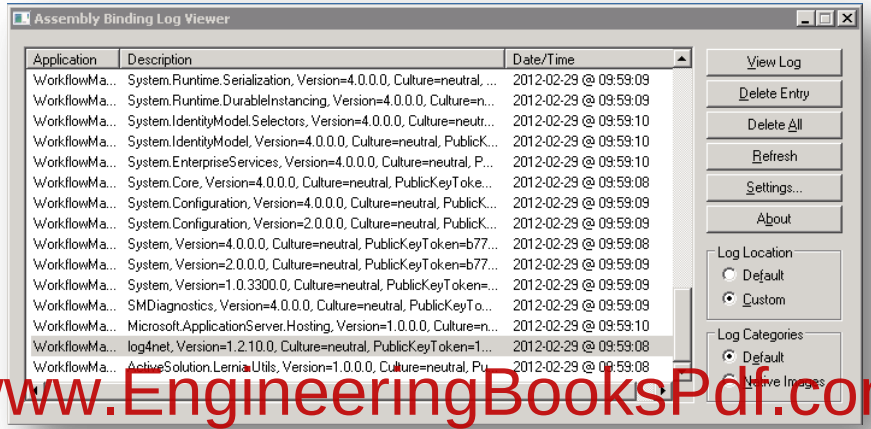





IT/Dev CONNECTIONS

# Assembly Loading Diagnostics

- **Fuslogvw** (in Windows/.NET SDK) can log all assembly bind failures / successes
  - Answers the question – why is my assembly loaded and where is it loaded from?



# Summary

- **Add application logs**
- Configure IIS logs and failed request tracing
- Server-side
  - Check Event Viewer
  - Check logs
- Client-side (in case exception is only see in client)
  - Check network traffic
- If exception is not logged, consider collecting a dump
  - Configure DebugDiag
  - Wait for it…
  - Analyze dump with Visual Studio
- If it is not your code, decompile it and find the bug

# W3WP CRASHES

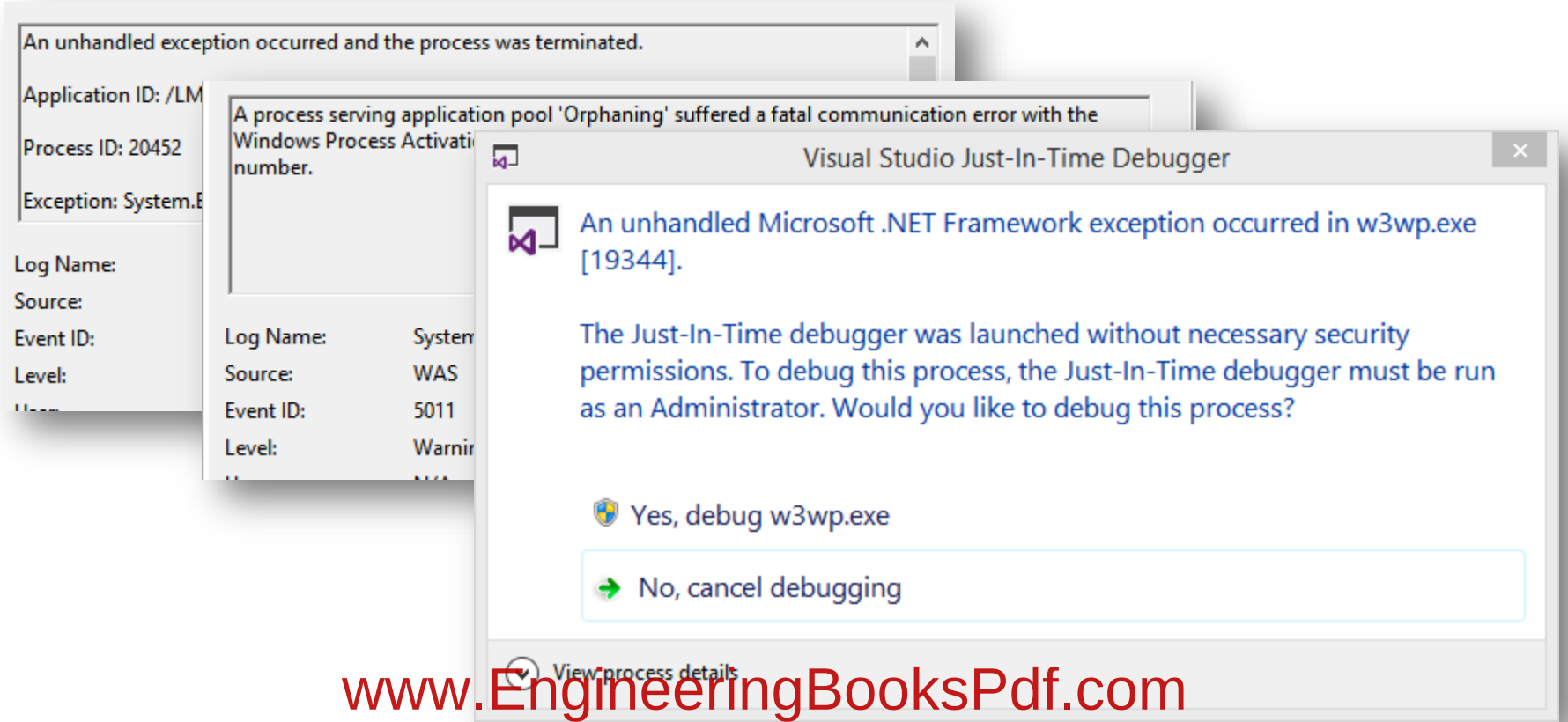
# Troubleshooting Tools

- Exceptions gone wild will cause processes to terminate and crash
- Logs
  - Event Viewer
  - Application logs (maybe)
- Crash dumps

# Why Stuff Crash?

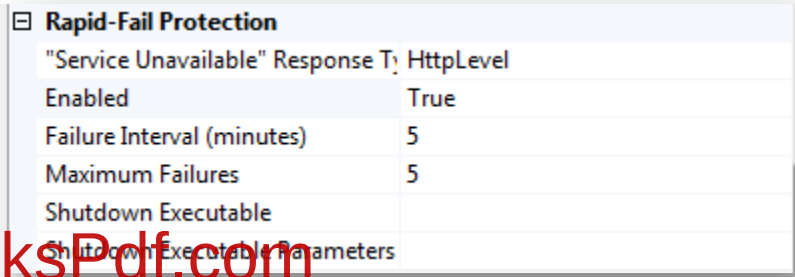
- Stack overflow
- Out-of-memory
- Unhandled exceptions in threads not monitored by ASP.NET
  - Thread objects
  - ThreadPool threads
  - Tasks - .NET 4 only (fixed in .NET 4.5)
  - Finalizer thread
- Native stuff
  - Unhandled exceptions in COM
  - Native/Managed heap corruption

# How Do You Know It Crashed?



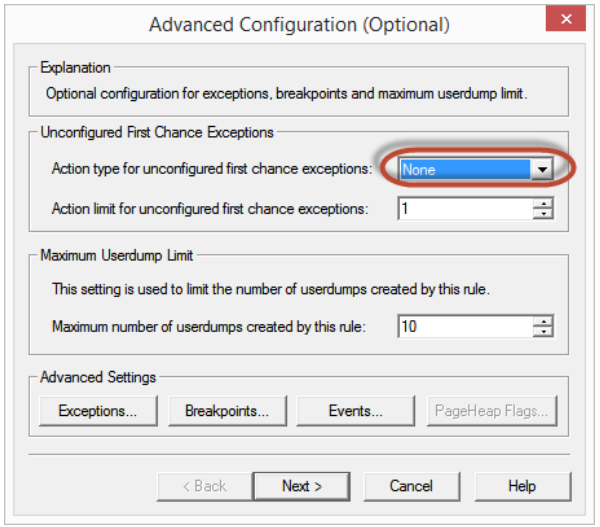
# Rapid Crash Cycle? IIS Rapid Fail Protection

- Rapid Fail Protection (RFP) occurs when app pool fails (crashes) X times in Y minutes (default is 5 crashes in 5 min.)
- Application pool is stopped, resulting in HTTP 503 responses
- Check the **System** event log for messages from **WAS**



# Collecting Crash Dumps with DebugDiag

- Same as collecting dumps for exceptions
- For crashes (second chance exceptions), leave it blank



# Summary

- Crashes are specific cases of exceptions
- We use the same techniques as with exceptions
- Usually a crash doesn't appear in application logs
- Configure dump collection for second chance exceptions

# SLOW RESPONSE TIME / HANGS

# Troubleshooting Tools

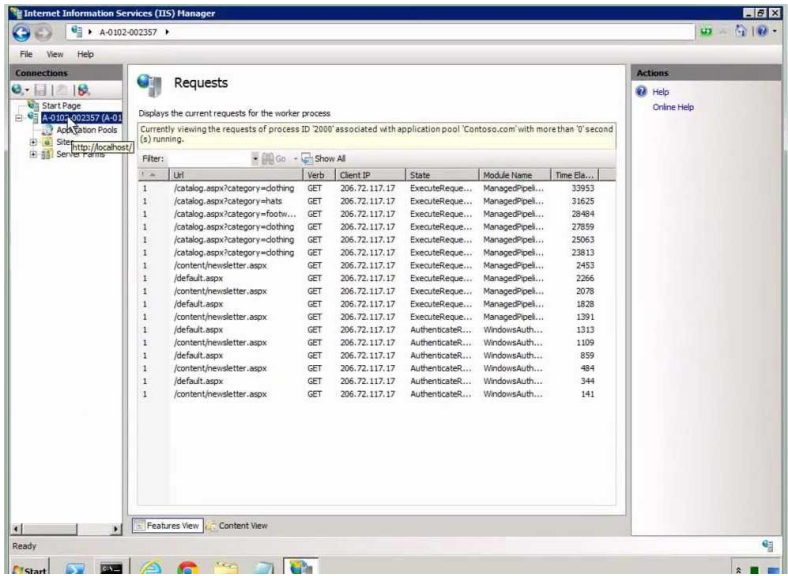
- IIS and application logs (mainly those containing timing)
- Performance counters
- Process Explorer
- Profiling tools
  - Visual Studio Profilers
  - Others (ANTS, dotTrace)
- Hang dumps
  - ProcDump
  - WinDbg
- Network sniffers

# Slow Response Time vs. Hang

- Slow response time = requests take long time to complete
- Hang = requests never complete or timeout
- Slow response time occurs due to:
  - Saturated CPU
  - Lock contention
  - Resource and I/O blocking (network, disk, database)
  - Many threads leading to context switches
  - Frequent GC
- Hang occurs due to:
  - Dead lock
  - Infinite loops
  - Same reasons of slow response, but on a larger scale

# Real Time Request View in IIS

- Runtime Status and Control API (RSCA)
- Shows currently executing:
  - Application Pools
  - Requests
- Exposed via
  - IIS admin tool
  - Programmatically via WMI
- Look out for requests still running, even after the client timed out



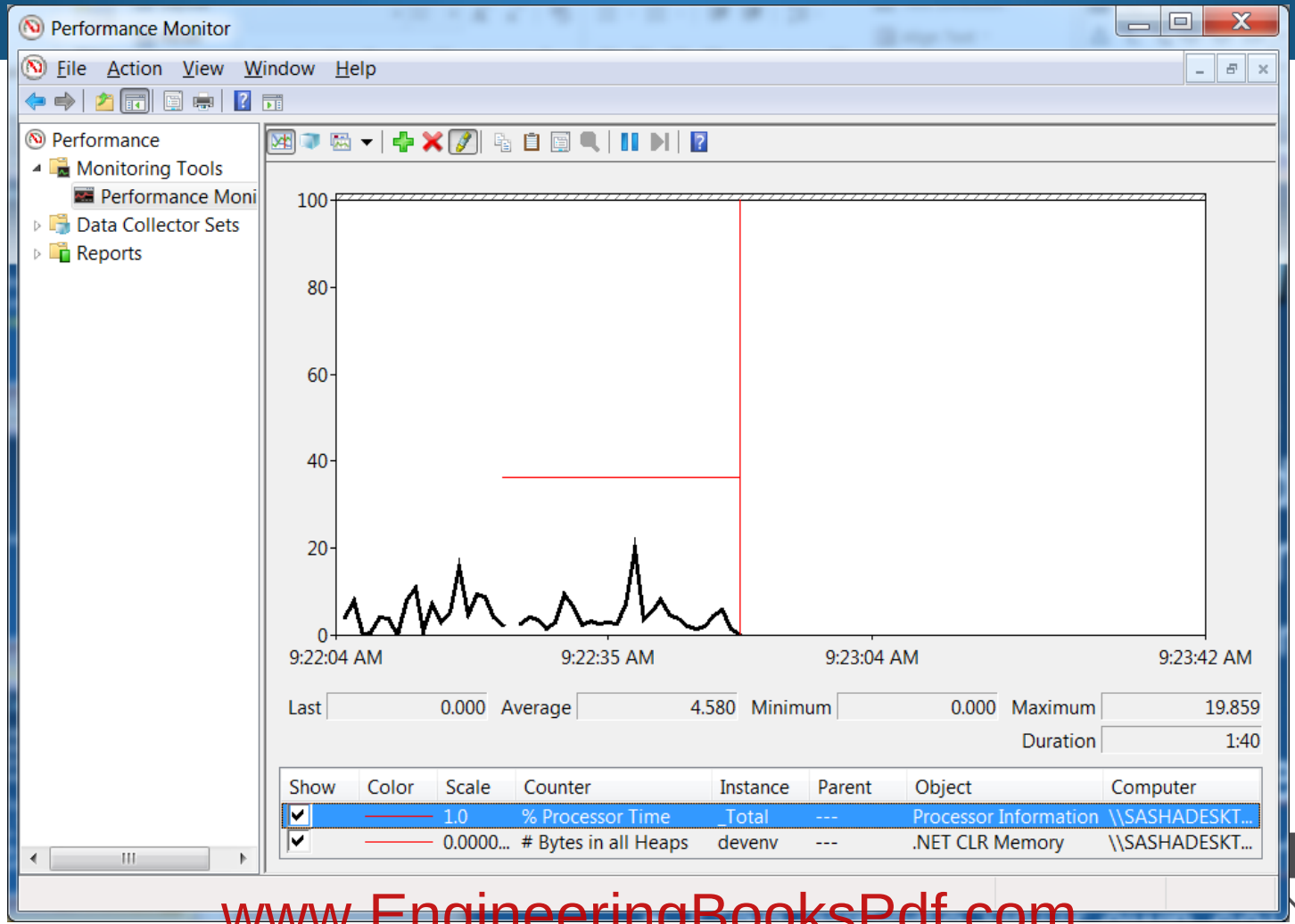
# What can we Learn from IIS?

- Logs
  - Check response "time-taken"
  - Identifying slow-to-respond URLs
  - Is it a single URL on all URLs during a specific time?
- FREB
  - Check latency of each module
  - Find the ones taking "too much"
  - Start your debug there
- RSCA
  - Locate infinite requests – may imply a dump is needed

# Performance Counters

# Performance Counters

- A set of numeric data exposed by Windows or by individual applications
  - Organized into **Categories**, **Instances**, and **Counters**
  - Example: Process(w3wp.exe)\Private Bytes
- Read with the built-in Performance Monitor MMC snap-in (**perfmon.exe**)
- Use Perfmon to record selected counters
- Expose your own application-level counters with *System.Diagnostics.PerformanceCounter*



# Performance Counters

- Many different performance counters can be useful for specific troubleshooting situations
- Web servers do not always benefit from adding more resources
- Adding a new server relieves resource problems, increasing the capacity of the web application that the servers are hosting
- A baseline is an important performance management tool

# System Resources Counters

## CPU and Memory Counters

- Processor\% Processor Time
- Processor\% Privileged Time
- Processor\% Interrupt Time
- System\Processor Queue Length
- System\Context Switches/sec
- Memory\Page Reads/sec

# System Resources Counters

## Network and Disk Counters

- Network Interface\Bytes Total/sec
- Network Interface\Bytes Received/sec
- Network Interface\Bytes Sent/sec
- PhysicalDisk\Avg. Disk Queue Length
- PhysicalDisk\Avg. Disk Read Queue Length
- PhysicalDisk\Avg. Disk Write Queue Length
- PhysicalDisk\Avg. Disk sec/Read
- PhysicalDisk\Avg. Disk sec/Transfer

# Managed Code Counters

## Memory Counters

- Process\Private Bytes
- .NET CLR Memory\% Time in GC
- .NET CLR Memory\# Bytes in all Heaps
- .NET CLR Memory\# Gen N Collections
- .NET CLR Memory\# of Pinned Objects
- .NET CLR Memory\Large Object Heap Size

# Managed Code Counters

## Threads and Locks

- Threading
  - Thread\% Processor Time
  - Thread\Context Switches/sec
  - Thread\Thread State
  - .NET CLR LocksAndThreads\# of current physical Threads
- Contention
  - .NET CLR LocksAndThreads\Contention Rate/sec
  - .NET CLR LocksAndThreads\Current Queue Length

# Managed Code Counters

## Other Counters

- Exceptions
  - .NET CLR Exceptions\# of Exceps Thrown / sec
- JIT
  - .NET CLR JIT\# of Methods JITted
  - .NET CLR JIT\% Time in Jit
- Code Access Security
  - .NET CLR Security\Total RunTime Checks
  - .NET CLR Security\Stack Walk Depth

# Managed Code Counters

## ASP.NET Counters

- Worker Process
  - ASP.NET\Worker Process Restarts
- Throughput
  - ASP.NET\Requests Current
  - ASP.NET Applications\Requests Executing
  - ASP.NET Applications\Pipeline Instance Count
  - Http Service Request Queues\CurrentQueueSize
  - ASP.NET Applications\Requests Timed Out
  - ASP.NET Applications\Requests/Sec
- Response time / latency
  - ASP.NET\Request Execution Time

# Introduction to ETW

- Integrated into Windows Desktop and Server
- Used by Microsoft (.NET, ASP.NET, IIS, …)
  - Your data side-by-side (by time, activity id)
- Wicked fast (kernel-level buffers)
- Semantically rich (time, stack, custom)
- Standardized tooling support

# ETW Scenarios

- Profile applications or the system itself (sampling)
- Obtain system-wide performance statistics:
  - Interrupts, DPCs, CPU utilization
  - By-process, by-thread, by-module statistics
- Obtain interesting CLR statistics:
  - JIT information and timings
  - GC events, pause times, memory reclamation
- Capture stack traces for interesting events:
  - Memory allocation stacks for applications/drivers
  - Hard page faults, direct disk/file operations

# ETW Tools

- **xperf.exe**: Command-line tool for ETW capturing and processing
  http://www.microsoft.com/en-US/download/details.aspx?id=39982
- **PerfView.exe**: Visual tool for capturing and recording ETW events from managed providers and the CLR
  http://www.microsoft.com/en-us/download/details.aspx?id=28567
- **wpr.exe**: Command-line and GUI for end users
- **wpa.exe**: Visual trace analysis tool
- WPR/WPA were added in the the Windows 8 SDK

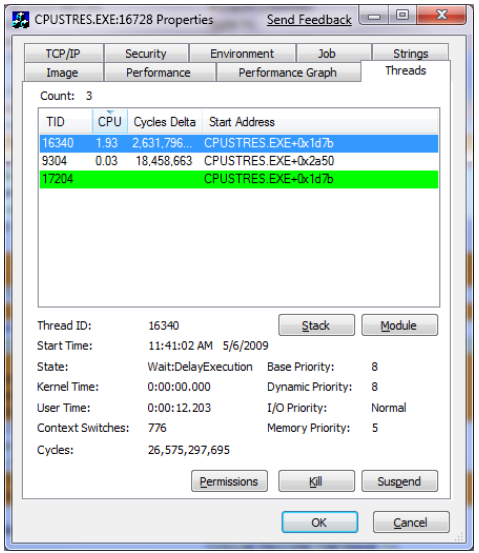
# Process Explorer

# Process Explorer

- Process Explorer is a Task Manager replacement
  - You can literally replace Task Manager with Options->Replace Task Manager
- Hide-when-minimized to always have it handy
  - Hover the mouse to see a tooltip showing the process consuming the most CPU
- Open System Information graph to see CPU usage history
  - Graphs are time stamped with hover showing biggest consumer at point in time
  - Also includes other activity such as I/O, kernel memory limits
- Download from:
  - https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx

# Viewing Threads

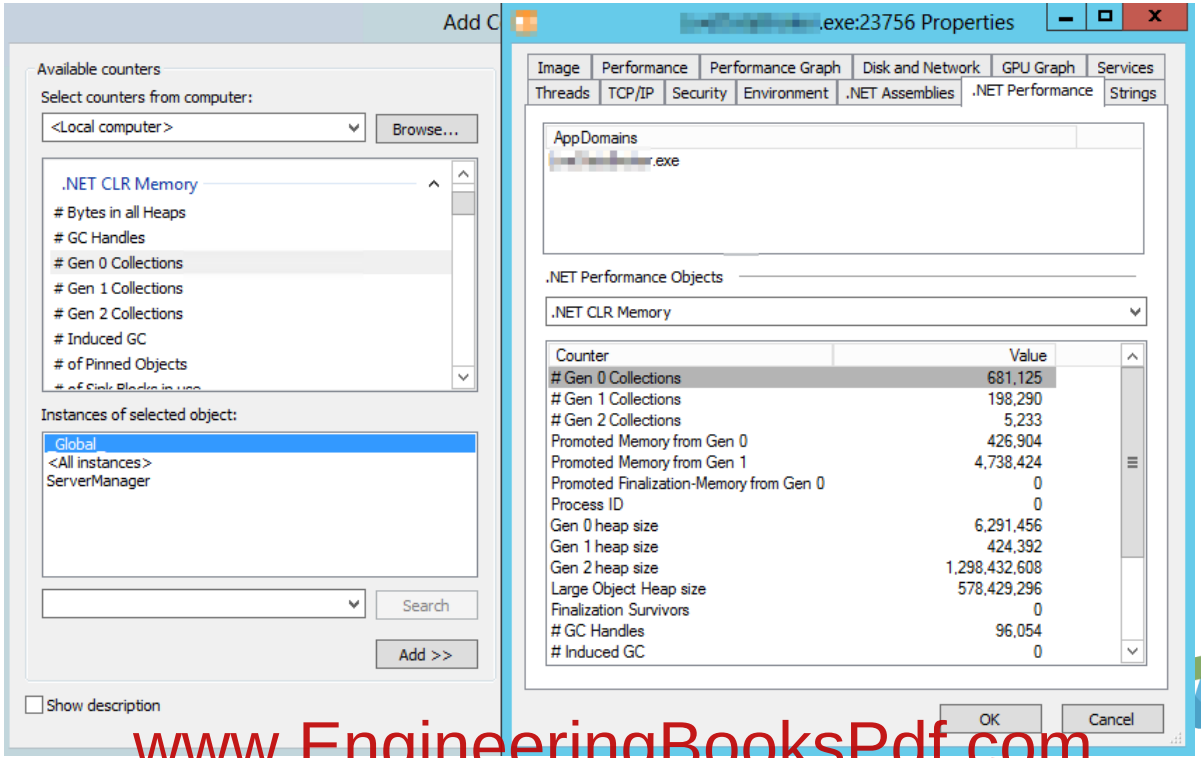
- Task Manager doesn't show thread details within a process
- Process Explorer does on "Threads" tab
- Displays thread details such as ID, CPU usage, start time, state, priority
- Click Stack to explore a stack (also applies to managed code)



# Thread Start Functions and Symbol Information

- Process Explorer can map the addresses within a module to the names of functions
  - This can help identify which component within a process is responsible for CPU usage
- Configure Process Explorer's symbol engine:
  - Download the latest Debugging Tools for Windows from Microsoft
  - Point at the Microsoft public symbol server (or internal symbol server)

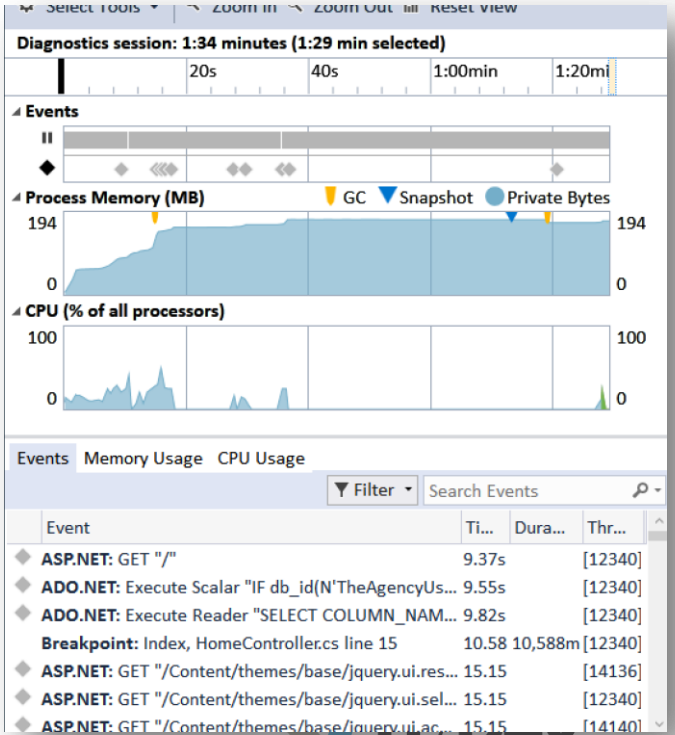
# Viewing Performance Counters With Process Explorer



# Profilers

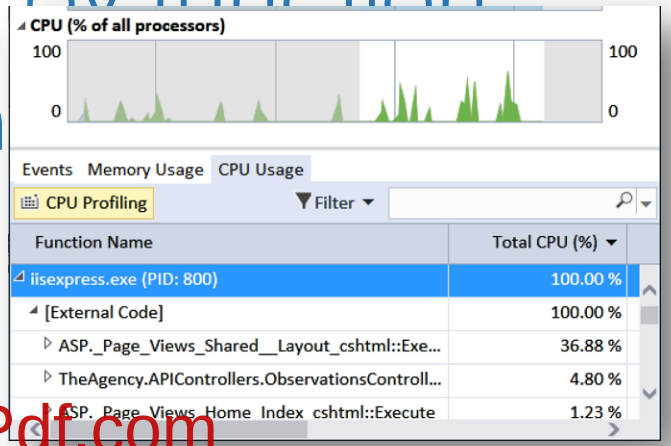
# Diagnostics Tool Window

- Just hit F5
- Memory and CPU usage graphs
- IntelliTrace UI is now part of the window
- Take memory snapshots and profile CPU usage
- Time sections of code with PerfTips



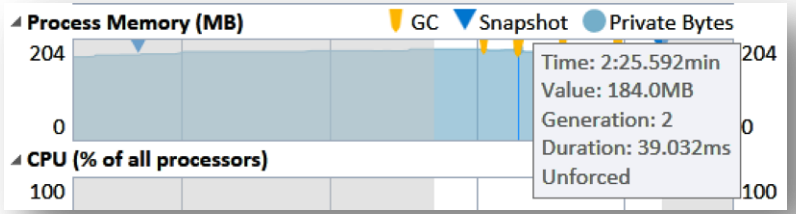
# Diagnostics Tool Window: CPU Usage

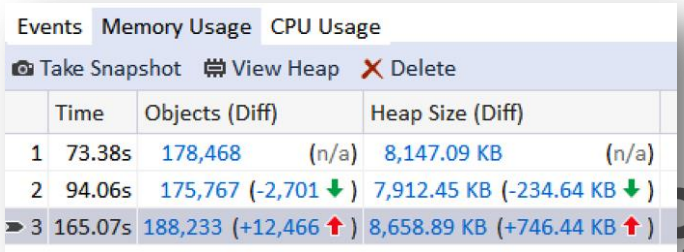
- Shows CPU utilization across all cores
- Turn on CPU profiling while debugging to get usage breakdown by function
- Available after debuggin session has stopped



# Diagnostics Tool Window: Memory Usage

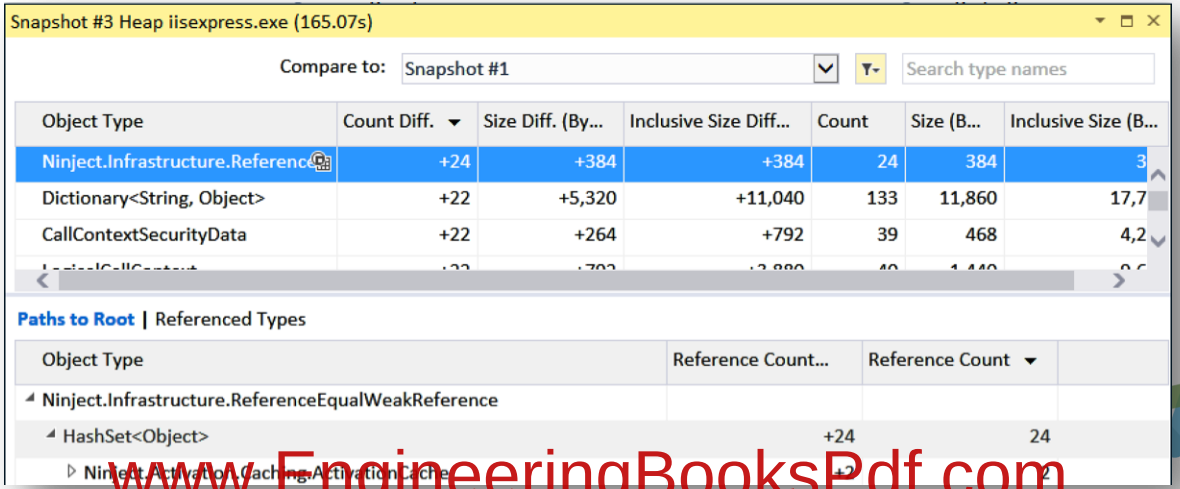
- Monitors memory usage of your app
- Can show native and managed heaps
- Take snapshots of your memory
- Ignores time spent in breakpoints
- Informs you when GC was activated and why



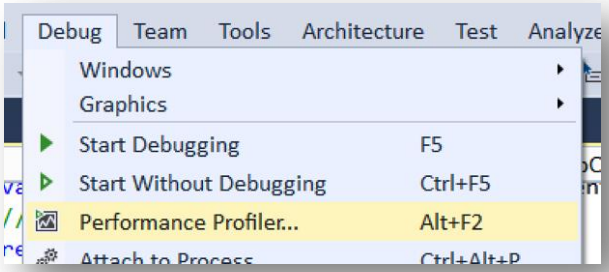
# Diagnostics Tool Window: Memory Usage

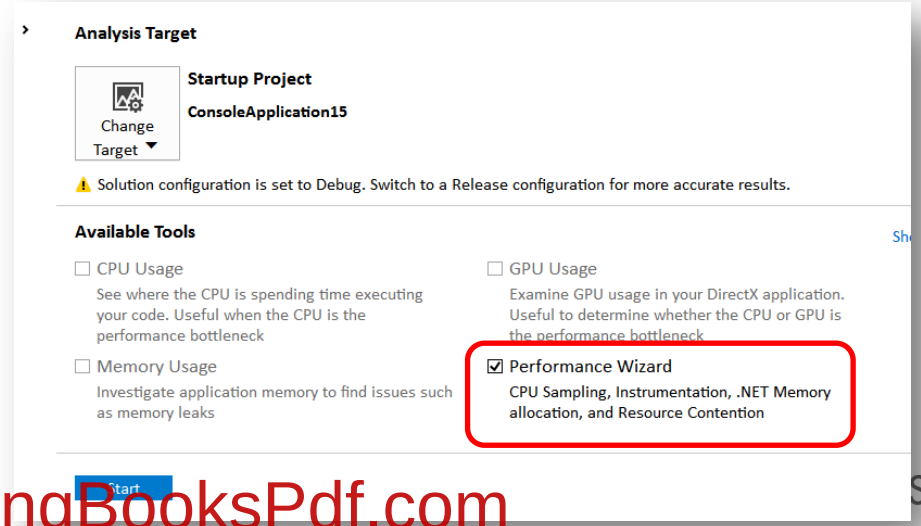
- Use memory snapshots to inspect the heap and to find memory leaks
  - Watch which objects were added, and their size
  - Track object paths to its root



# Who Moved my Profilers?

- Under the *Analyze* menu in VS 2010
- Gone into hiding in VS 2013
- Still in hiding in VS 2015



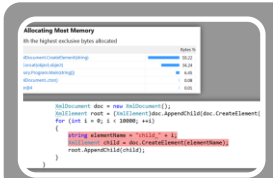



# The Visual Studio Profilers

- Same good old profilers
- Can run as stand-alone from the command-line

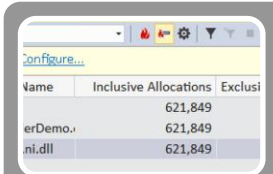

## Sampling

- CPU-bound apps, very low overhead
- Full program stacks (including all system DLLs)




## Instrumentation

- I/O-bound apps, CPU-bound apps, higher overhead
- More detailed timing data, limited stacks (just my code)




## Allocations

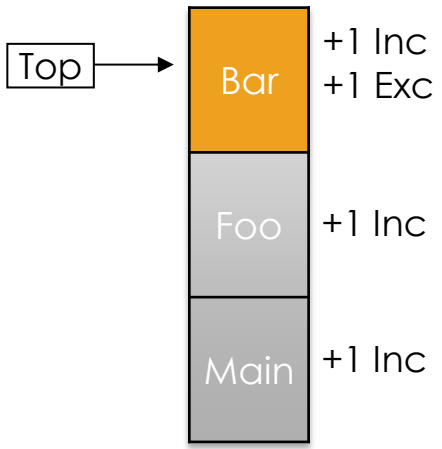
- Details on who allocated and what
- Managed code only

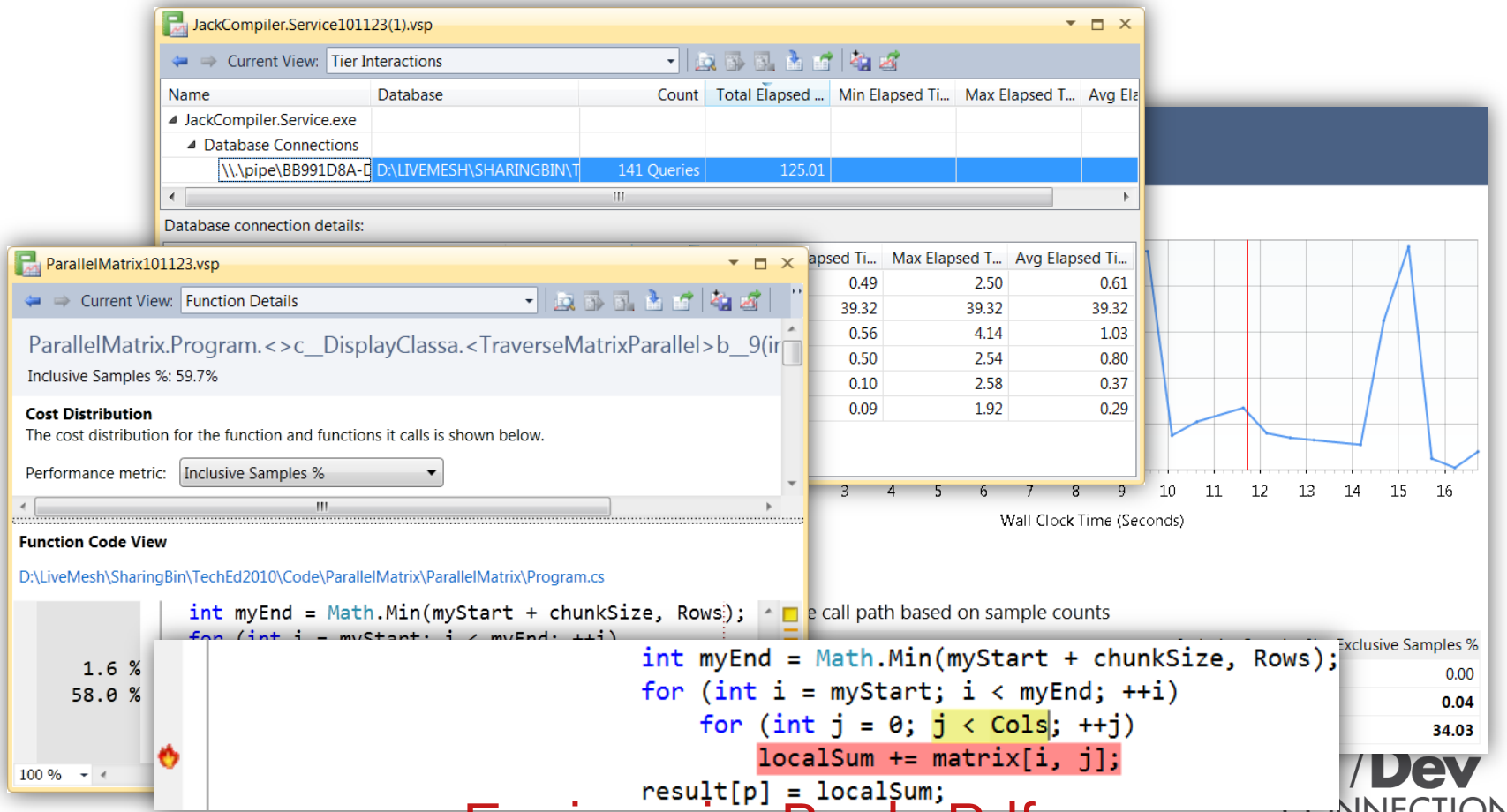
# Sampling Profiler

- Periodically interrupt the application
- Walk the application's stack
  - Record the frames, no symbol resolution yet
  - Very fast, small intrusion, little effect on profiled app

# Analyzing the Data

- Exclusive samples
  - Function was on the top of the stack
  - Function is doing a lot of individual work
- Inclusive samples
  - Function was on the stack (but not the top)
  - Function causes a lot of work to be done





# Interpreting the Results

- Samples != Time
  - Blocked functions don't get samples
  - There may be statistical errors (an "evasive" function that never shows up during a sample)
- Very long runs are not necessary
  - Long runs = more noise = less clarity
- Make sure you have debugging symbols

# Instrumentation Profiler

- The profiler ***instruments*** the binary before it's launched
  - Emits markers that record function execution times and counts
  - In other profilers, can work at the line-level as well – but very expensive

```
void foo()
{
    FUNC_ENTER(foo);
    // do some work
    CALL_ENTER(ExtCall);
    // call another function
    ExtCall();
    CALL_EXIT(ExtCall);
    // do some more work
    FUNC_EXIT(foo);
}
```

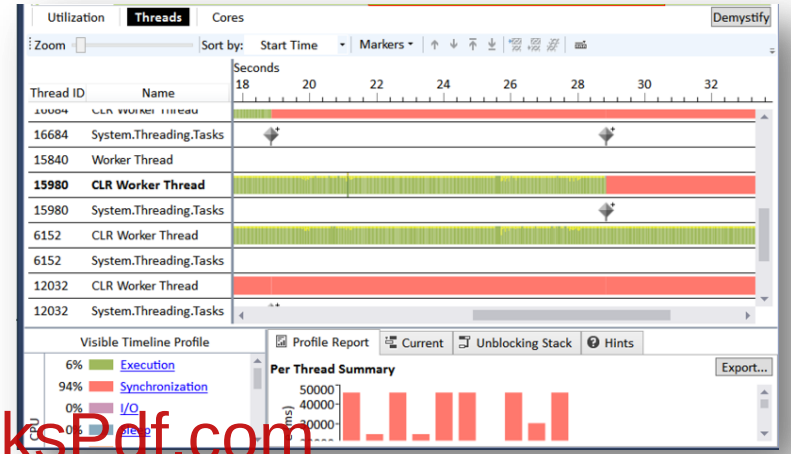
# Instrumentation Analysis

- More detailed performance data
  - Number of calls
  - Actual Time (probe overhead is subtracted)
- Elapsed time
  - Raw time spent in the function (wall clock time)
- Application time
  - Probes are marked when kernel transitions occur between two probes
  - That time is discounted in Application time

# Concurrency Visualizer

- Analyze the application's concurrency characteristics
  - CPU utilization
  - Thread blocking and migration
  - Resource contention
- Downloadable as a Visual Studio Extension

http://bit.ly/2a5kbay



# 3rd-Party Profiles

- JetBrains dotTrace
  - https://www.jetbrains.com/profiler
- Redgate ANTS performance profiler
  - http://www.red-gate.com/products/dotnet-development/ants-performance-profiler
- Telerik JustTrace
  - http://www.telerik.com/products/memory-performance-profiler.aspx

# Hang Dumps

# When to Collect Dumps

- Hang dumps with IIS
  - Configure IIS for Orphaning
  - Use **Procdump.exe** to collect the dump
- Hang dumps with DebugDiag
  - Create a **Performance** rule in DebugDiag
  - Trigger by performance counter or HTTP response time
- Manual collection
  - Open **Task Manager**, right-click and choose **Create dump file**

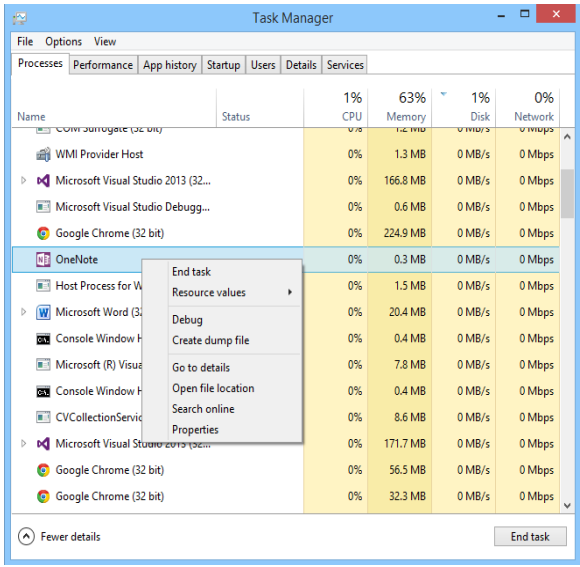
# Sysinternals Procdump

- Take dumps easily, anywhere, and with no pun intended

```
Procdump -ma -e MyApp.exe
Procdump -h -x C:\temp\myapp.exe
Procdump -c 90 -n 3 -s 5 MyApp.exe
Procdump -e 1 1234
```

# Generating a Hang Dump

- Right-click process and choose **Create Dump File**
  - For 32-bit, open %WINDIR%\SysWOW64\taskmgr.exe
- Dump created at %LOCALAPPDATA%\Temp



# Opening Dump Files

- You can still use Visual Studio
  - Inspect managed threads to see what each is doing
  - Try to identify threads waiting on locks
- Or…
- Use WinDbg and SOS.dll

# Enter:

## WinDbg

- Lightweight GUI debugger
- Super-scriptable
- Super-extensible
- Knows nothing about .NET ☹

## SOS

- WinDbg extension for .NET
- Ships with .NET Framework
- Or on the symbol server (As of CLR 4.0)
- Knows all about .NET ☺

Download instructions:
https://msdn.microsoft.com/library/windows/hardware/ff551063

# Symbols for Microsoft Binaries

- Microsoft has a public symbol server with PDB files for Microsoft binaries
- Configure _NT_SYMBOL_PATH environment variable

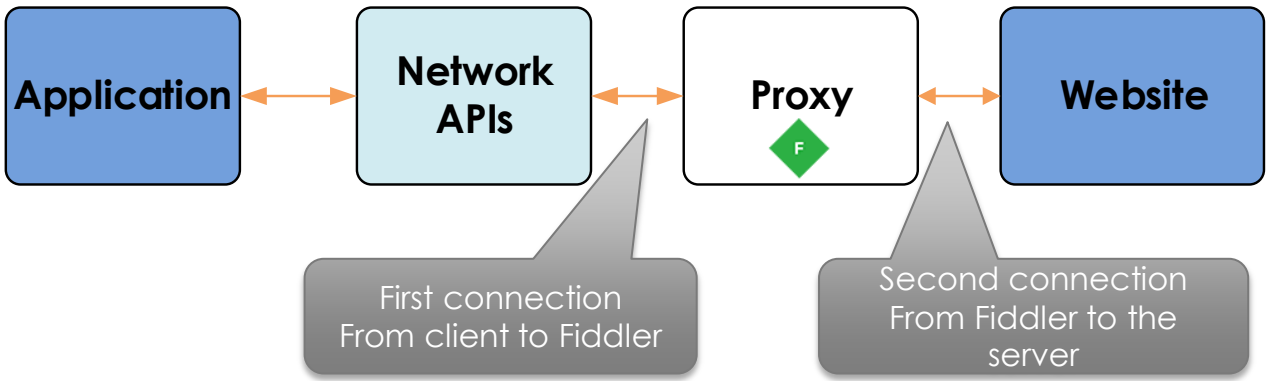
```
setx _NT_SYMBOL_PATH
srv*C:\symbols*http://msdl.microsoft.com/download/symbols
```

# WinDbg Cheat Sheet

- Loading SOS / SOSEX
  - .loadby sos clr
  - .load sosex
  - .chain
- Threads and stacks
  - !threads
  - ~Ns (where N is the thread number)
  - !clrstack
- Locks
  - !syncblk
  - !mlocks (sosex)
  - !mwait (sosex)
  - !dlk (sosex)

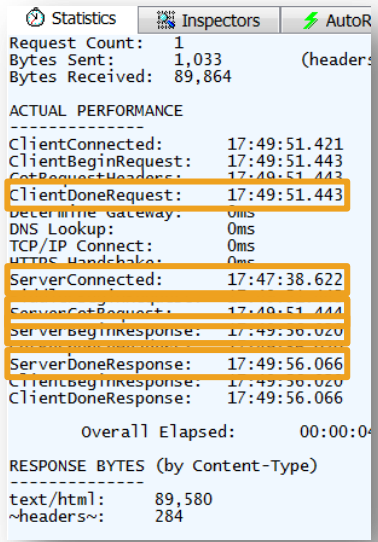
# Network Sniffers

# Fiddler Reminder: Understanding the Connection



# Fiddler Statistics are Important

- Single page statistics give info on performance
  - Client processing
  - Server processing
  - Network latency
- Server time:
  **ServerBeginResponse – ServerGotRequest**
- Upload time:
  **ServerGotRequest – ClientDoneRequest**
- Download time:
  **ServerDoneResponse – ServerBeginResponse**
- Watch out for misleading connection reuse
- Add timer columns instead of calculating



# Summary

- There are many tools to diagnose slow response time and hangs, because there are many reasons for this behavior
- When possible, run profilers while developing / testing
- Performance Counters are very useful, if you know how to "decipher" them
- When your production app hangs, use dumps to see what each thread is doing

# HIGH MEMORY USAGE

# Troubleshooting Tools

- Memory allocation profilers
  - Visual Studio Allocation profiler
  - ANTS / dotMemory / JustTrace
- Dump analysis
  - WinDbg
  - Visual Studio 2015

# High Memory Usage and OOM

- What are you leaking? (native, .NET, assemblies)
- Take multiple dumps when memory is raising and compare
- Check the finalizer
- Check the large object heap (LOH)
- Try to group "leaking" objects
- Figure out why they are sticking around (rooted)

# What are we Leaking?

- Caching and session state
- "Unexpected roots"
- Blocked finalizer
- DataSet serialization
- Large viewstate
- Assembly leaks with XmlSerializer
- Pinned objects

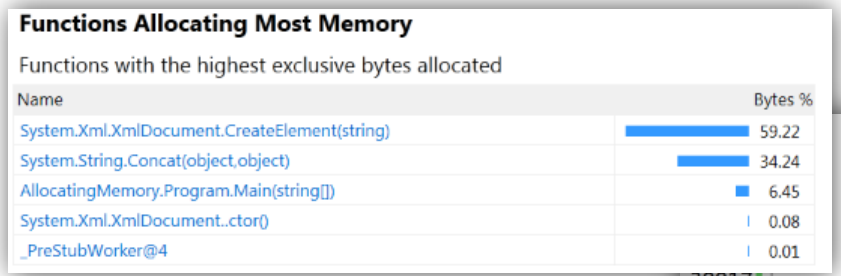
# Memory Allocation Profilers

# Memory Allocation Profiling

- Leaks are caused by the failure to release memory allocated beyond typical short term allocations and caching
- Memory allocations incur a significant cost
  - The allocations are cheap, but the GC isn't!
  - You won't always see the cost at the source, because the allocating function runs quickly
- Profiling an application for excessive allocations may be more important than CPU time
  - Another aspect is diagnosing memory leaks or sources of excess memory consumption

# Visual Studio Allocation Profiling

- Identifies the locations making the most allocations, and lists the types and allocation counts



**Functions Allocating Most Memory**

Functions with the highest exclusive bytes allocated

| Name | Bytes % |
|---|---|
| System.Xml.XmlDocument.CreateElement(string) | 59.22 |
| System.String.Concat(object,object) | 34.24 |
| AllocatingMemory.Program.Main(string[]) | 6.45 |
| System.Xml.XmlDocument..ctor() | 0.08 |
| _PreStubWorker@4 | 0.01 |

```
XmlDocument doc = new XmlDocument();
XmlElement root = (XmlElement)doc.AppendChild(doc.CreateElement("root"));
for (int i = 0; i < 10000; ++i)
{
    string elementName = "child_" + i;
    XmlElement child = doc.CreateElement(elementName);
    root.AppendChild(child);
}
```

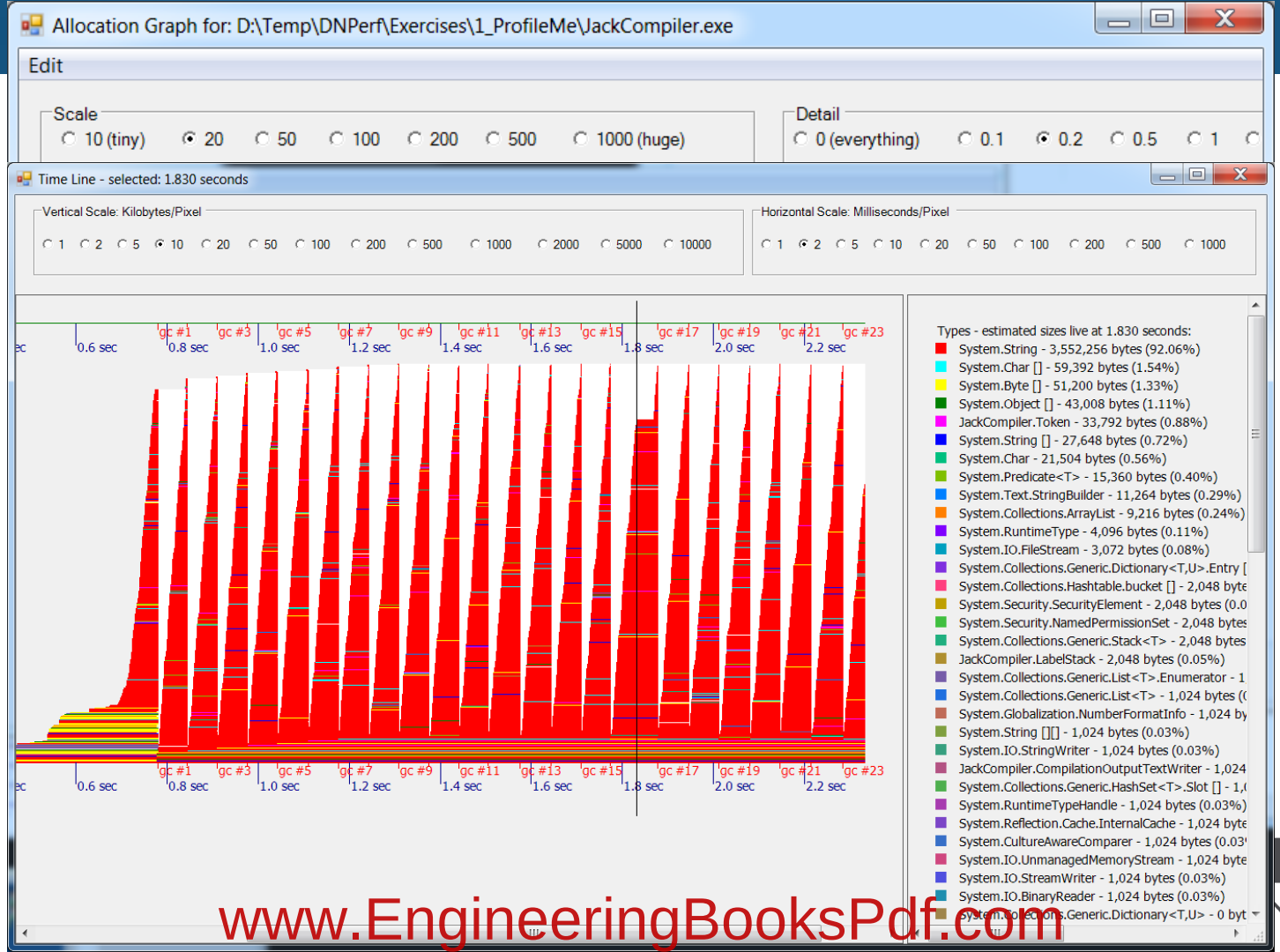
30017

# Other Memory Profilers

- Features
  - Commercial-grade profiler for analyzing memory allocations and leaks
  - Runs the application, captures heap snapshots, and analyzes object retention information
  - Can point out the source of a leak
  - Can filter out displayed objects by a variety of roots
- Redgate ANTS Memory Profiler
- JetBrains dotMemory
- Telerik JustTrace

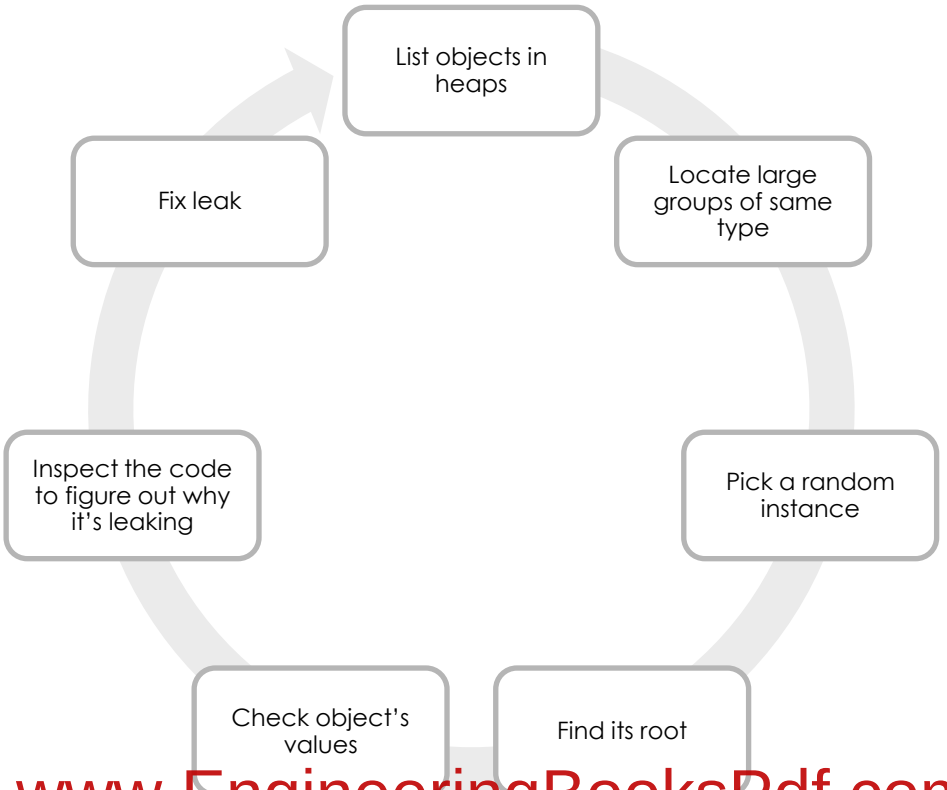
# CLR Profiler

- An intrusive tool that tracks memory allocations:
  - Allocation details
  - Allocations graphs
  - Breakdown of the managed heap
- May cause a significant application slowdown – every allocation is tracked
- Biggest advantage: It's free, very small, and comes with sources
- Download from: http://clrprofiler.codeplex.com



# Diagnosing Dumps for Leaks with WinDbg

# State Machine for Memory Diagnostics

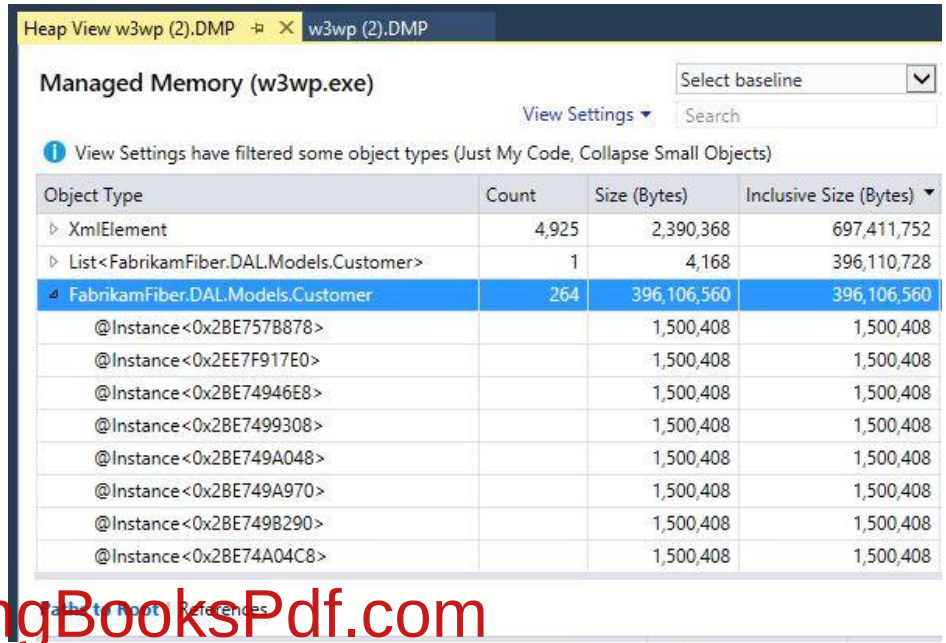


# WinDbg Commands for Memory

- !DumpHeap – Look at all the objects in your process
- !DumpDynamicAssemblies – List dynamically loaded assemblies
- !GCRoot – Find what is referencing the object
- !GCWhere – Tells you if the runtime has tried to collect it
- Show object values
  - !DumpObj – Classes
  - !DumpVC – Structs
  - !DumpArray – Arrays
  - !DumpClass – Static values
- !ObjSize – Calculates total size of an object and its children

# Memory Dump Analysis with Visual Studio 2015

- "Debug Managed Memory"
- Enterprise only

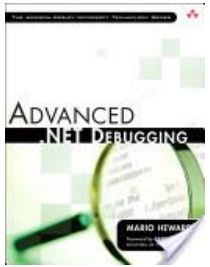


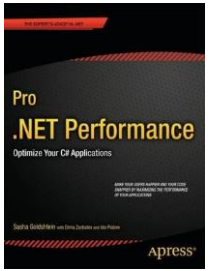
# Summary

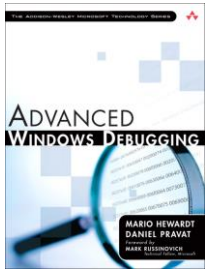
- Fixing memory leaks is an ongoing process
- Pick your fights
  - Many small objects
  - Small amount of very large objects (usually arrays)
- Remember that a single rooted object can point to many small objects
- Use professional memory profilers when possible
  - Easier to use than WinDbg
  - Find roots faster than with WinDbg

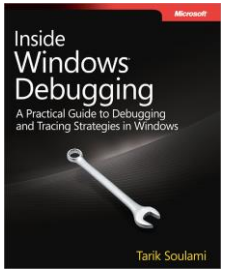
# Resources

- Great blogs on troubleshooting and web apps
  - http://blogs.msdn.com/b/tess/
  - http://blogs.msdn.com/b/asiatech/
  - http://blogs.msdn.com/b/webdev/
- Read books




- Contact me
  - idof@sela.co.il
  - @idoFlatow

# Rate This Session Now!

Tell Us What You Thought of This Session

## Rate with Mobile App:

- Select the session from the Agenda or Speakers menus
- Select the Actions tab
- Click Rate Session


## Rate with Website:

Register at **www.devconnections.com/logintoratesession**

Go to **www.devconnections.com/ratesession**

Select this session from the list and rate it

#ITDevConnections



IT/Dev CONNECTIONS