https://s.sashag.net/prodsdd

Sasha Goldshtein
CTO, Sela Group

blog.sashag.net
@goldshtn

Setting Up a Production Monitoring and Diagnostic Environment

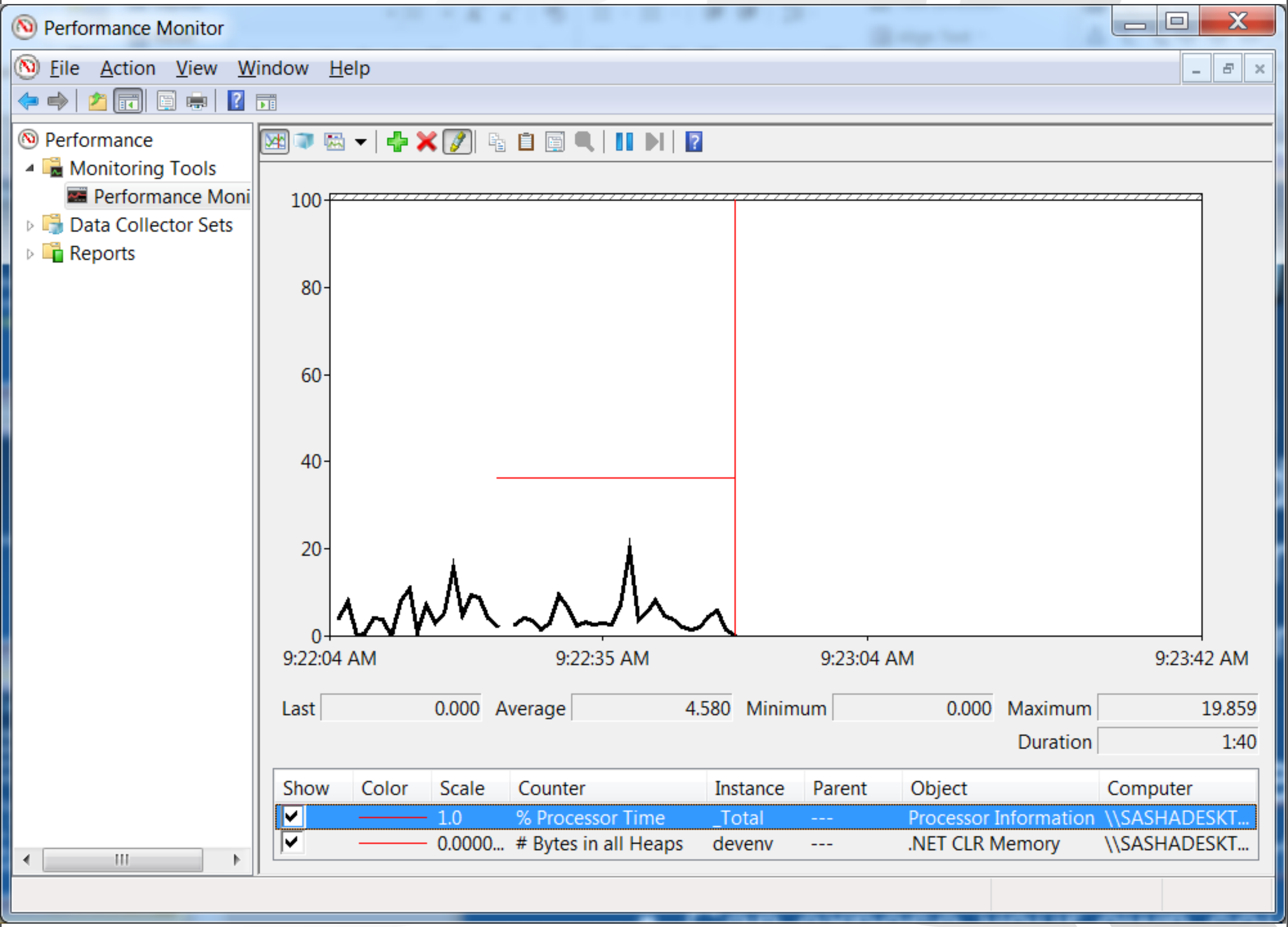
Agenda

- **Performance monitoring**
  Performance counters and alerts
  ETW, WPR, WPA, PerfView
- **Production debugging**
  IntelliTrace
  Dump files and dump analysis
- **“Automatic” debugging**
  CLRMD and CLRMDExt

Performance Counters

Performance Counters

- A set of numeric data exposed by Windows or by individual applications
  - Organized into *Categories*, *Instances*, and *Counters*
  - Example: `Process(Outlook.exe)\Private Bytes`
- Accessed using `System.Diagnostics`:
  - `PerformanceCounter`, `PerformanceCounterCategory`
  - Can expose your own counters as well
- Tools: **perfmon.exe**, **logman.exe**, **lodctr.exe**

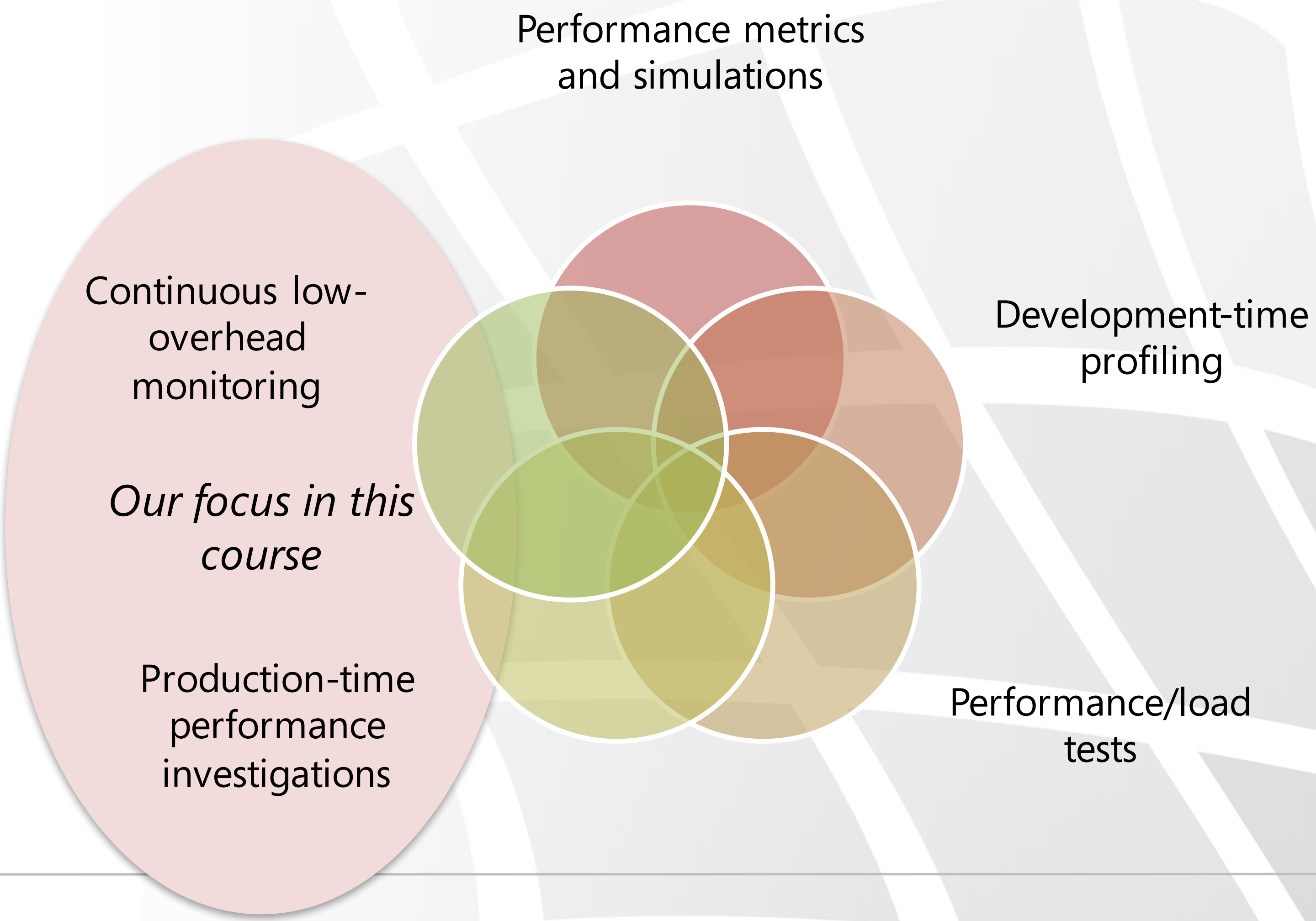


Last	0.000	Average	4.580	Minimum	0.000	Maximum	19.859
						Duration	1:40

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
✓	🟥	1.0	% Processor Time	_Total	---	Processor Information	\\SASHADESKT...
✓	🟥	0.0000...	# Bytes in all Heaps	devenv	---	.NET CLR Memory	\\SASHADESKT...

Introduction to ETW

Performance Monitoring Spectrum


Performance metrics and simulations
Development-time profiling
Performance/load tests
Continuous low-overhead monitoring
Our focus in this course
Production-time performance investigations

Problems with Traditional Profilers

Invasiveness

- Often requires restart or code injection

Overhead

- 2x slowdowns are not unheard of

Trace size

- Often not applicable for continuous monitoring for hours/days on end

Licensing costs

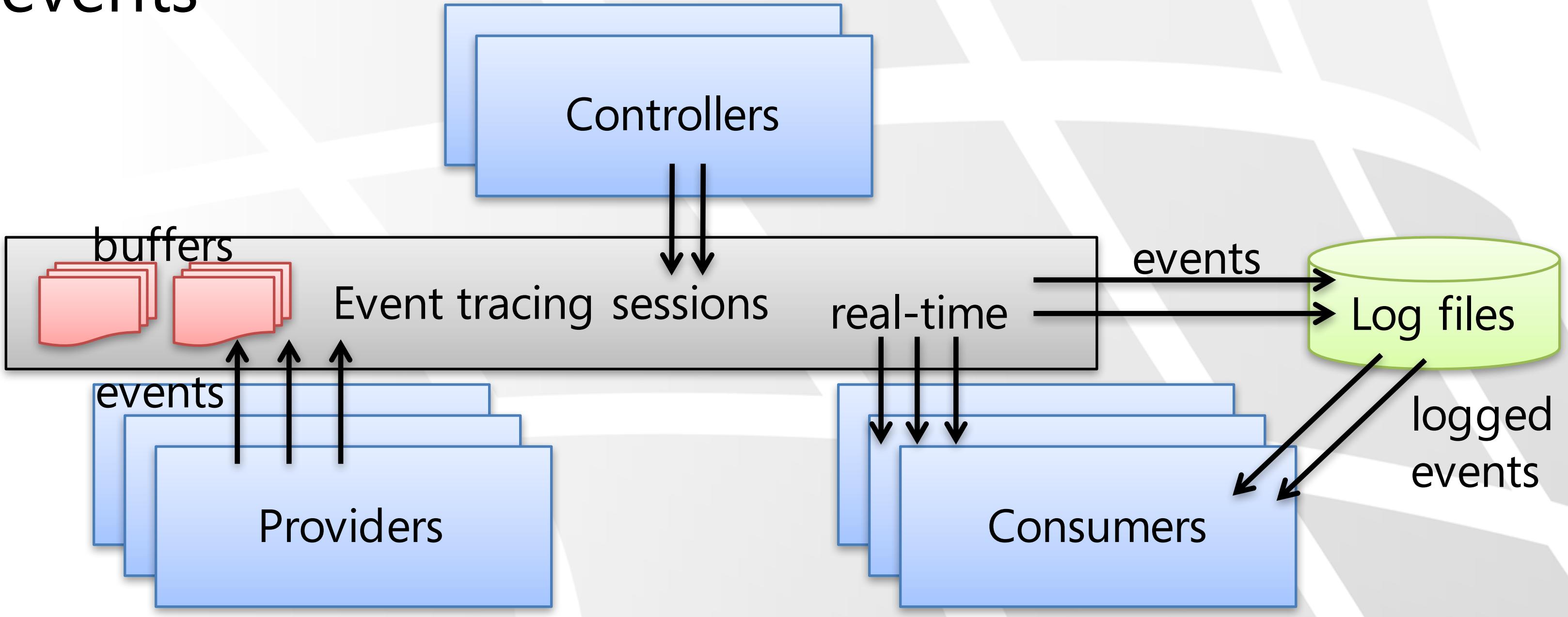
- Production mode or remote profiling mode not always available

Event Tracing for Windows

- High-performance facility for emitting 100K+ log events per second with rich payloads and stack trace support
- Used widely across Windows, .NET, drivers, services, third party components

ETW Participants

- A **provider** generates ETW events
- A **controller** starts and stops ETW collection
- A **consumer** logs, analyzes, or processes ETW events



Sample ETW Scenarios

Profile an app in sampling mode

Perform wait-time analysis

Log disk accesses including stacks

Log GC and JIT events

Log memory allocation statistics (.NET/C++)

Custom application event log

Trace Capturing and Analysis

ETW Tools

- **xperf.exe**: Command-line tool for ETW capturing and processing
- **wpr.exe**: Command-line and GUI for end users
- **wpa.exe**: Visual trace analysis tool
- **PerfView.exe**: Visual tool for capturing and recording ETW events from managed providers and the CLR
- **logman.exe**, **tracerpt.exe**: Built-in Windows tools for trace recording and formatting

Production Use

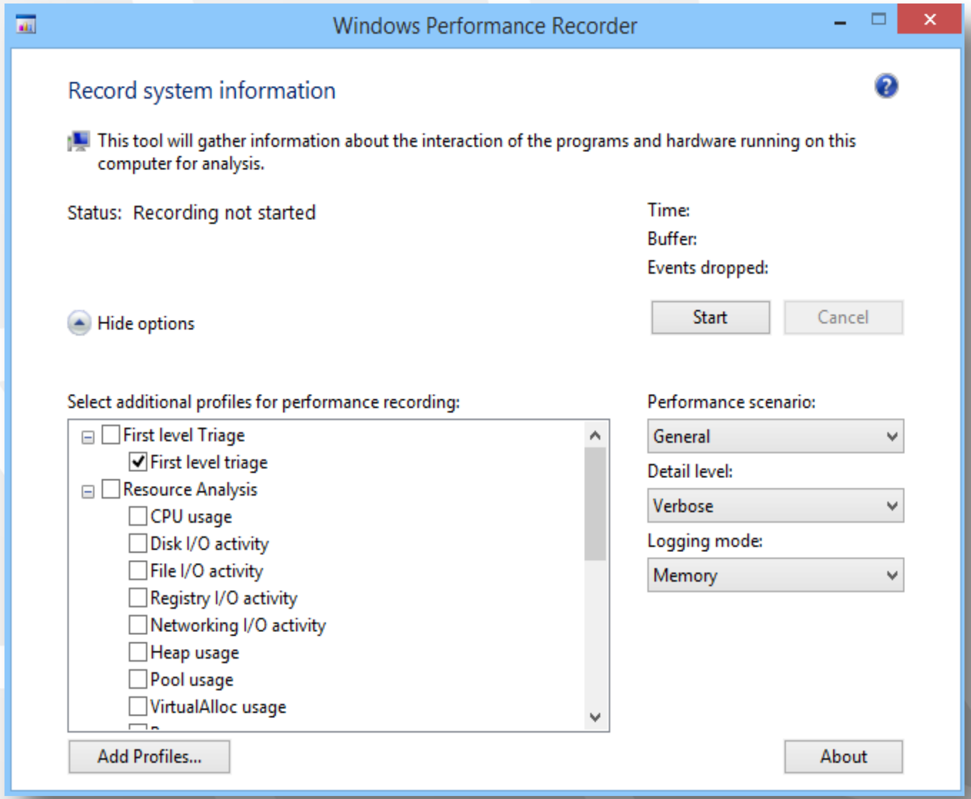
- All ETW tools are suitable for production use
- Some things to watch out for:
  - Choose event providers carefully to minimize the performance impact on the system
  - Capture to a circular log file to avoid running out of disk space
  - Set triggers to stop collection (and keep all preceding events) when a critical event occurs

Capturing a Trace

Xperf

```
xperf -on DiagEasy
...
xperf -d diag.etl
```

WPR


Windows Performance Recorder

Record system information

This tool will gather information about the interaction of the programs and hardware running on this computer for analysis.

Status: Recording not started

Time:
Buffer:
Events dropped:

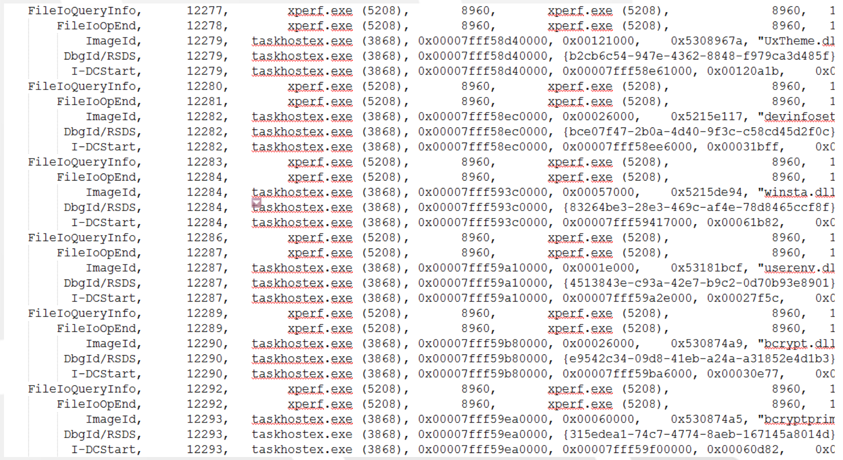
Select additional profiles for performance recording:

- [ ] First level Triage
  - [✓] First level triage
- [ ] Resource Analysis
  - [ ] CPU usage
  - [ ] Disk I/O activity
  - [ ] File I/O activity
  - [ ] Registry I/O activity
  - [ ] Networking I/O activity
  - [ ] Heap usage
  - [ ] Pool usage
  - [ ] VirtualAlloc usage

Performance scenario: General
Detail level: Verbose
Logging mode: Memory

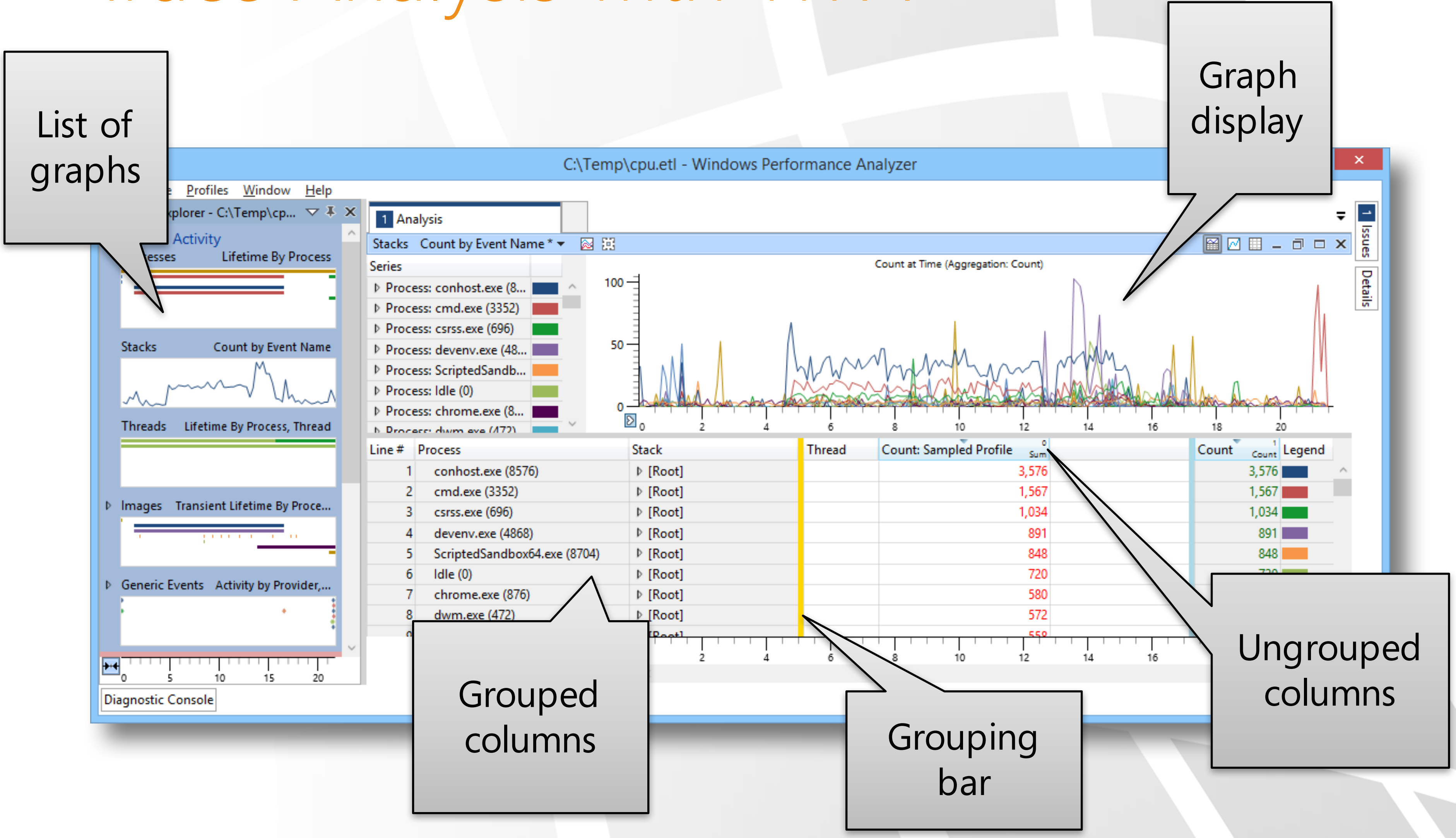
What’s In A Trace?

- A trace is a huge list of events
- Events have multiple columns (payload)
- Useless without additional processing



```
FileIoQueryInfo,      12277,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
    FileIoOpEnd,      12278,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
        ImageId,      12279,   taskhostex.exe (3868), 0x00007fff58d40000, 0x00121000,     0x5308967a, "UxTheme.dl
     DbgId/RSDS,      12279,   taskhostex.exe (3868), 0x00007fff58d40000, {b2cb6c54-947e-4362-8848-f979ca3d485f}
      I-DCStart,      12279,   taskhostex.exe (3868), 0x00007fff58d40000, 0x00007fff58e61000, 0x00120a1b,    0xC
FileIoQueryInfo,      12280,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
    FileIoOpEnd,      12281,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
        ImageId,      12282,   taskhostex.exe (3868), 0x00007fff58ec0000, 0x00026000,     0x5215e117, "devinfoset
     DbgId/RSDS,      12282,   taskhostex.exe (3868), 0x00007fff58ec0000, {bce07f47-2b0a-4d40-9f3c-c58cd45d2f0c}
      I-DCStart,      12282,   taskhostex.exe (3868), 0x00007fff58ec0000, 0x00007fff58ee6000, 0x00031bff,    0xC
FileIoQueryInfo,      12283,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
    FileIoOpEnd,      12284,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
        ImageId,      12284,   taskhostex.exe (3868), 0x00007fff593c0000, 0x00057000,     0x5215de94, "winsta.dll
     DbgId/RSDS,      12284,   taskhostex.exe (3868), 0x00007fff593c0000, {83264be3-28e3-469c-af4e-78d8465ccf8f}
      I-DCStart,      12284,   taskhostex.exe (3868), 0x00007fff593c0000, 0x00007fff59417000, 0x00061b82,    0xC
FileIoQueryInfo,      12286,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
    FileIoOpEnd,      12287,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
        ImageId,      12287,   taskhostex.exe (3868), 0x00007fff59a10000, 0x0001e000,     0x53181bcf, "userenv.dl
     DbgId/RSDS,      12287,   taskhostex.exe (3868), 0x00007fff59a10000, {4513843e-c93a-42e7-b9c2-0d70b93e8901}
      I-DCStart,      12287,   taskhostex.exe (3868), 0x00007fff59a10000, 0x00007fff59a2e000, 0x00027f5c,    0xC
FileIoQueryInfo,      12289,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
    FileIoOpEnd,      12289,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
        ImageId,      12290,   taskhostex.exe (3868), 0x00007fff59b80000, 0x00026000,     0x530874a9, "bcrypt.dll
     DbgId/RSDS,      12290,   taskhostex.exe (3868), 0x00007fff59b80000, {e9542c34-09d8-41eb-a24a-a31852e4d1b3}
      I-DCStart,      12290,   taskhostex.exe (3868), 0x00007fff59b80000, 0x00007fff59ba6000, 0x00030e77,    0xC
FileIoQueryInfo,      12292,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
    FileIoOpEnd,      12292,        xperf.exe (5208),          8960,        xperf.exe (5208),            8960,   1
        ImageId,      12293,   taskhostex.exe (3868), 0x00007fff59ea0000, 0x00060000,     0x530874a5, "bcryptprim
     DbgId/RSDS,      12293,   taskhostex.exe (3868), 0x00007fff59ea0000, {315edea1-74c7-4774-8aeb-167145a8014d}
      I-DCStart,      12293,   taskhostex.exe (3868), 0x00007fff59ea0000, 0x00007fff59f00000, 0x00060d82,    0xC
```

Trace Analysis with WPA



Series
- Process: conhost.exe (8... 🟦
- Process: cmd.exe (3352) 🟥
- Process: csrss.exe (696) 🟩
- Process: devenv.exe (48... 🟪
- Process: ScriptedSandb... 🟧
- Process: Idle (0) 🟩
- Process: chrome.exe (8... 🟪
- Process: dwm.exe (472) 🟦

Line #	Process	Stack	Thread	Count: Sampled Profile Sum	Count	Legend
1	conhost.exe (8576)	[Root]		3,576	3,576	🟦
2	cmd.exe (3352)	[Root]		1,567	1,567	🟥
3	csrss.exe (696)	[Root]		1,034	1,034	🟩
4	devenv.exe (4868)	[Root]		891	891	🟪
5	ScriptedSandbox64.exe (8704)	[Root]		848	848	🟧
6	Idle (0)	[Root]		720	720	🟩
7	chrome.exe (876)	[Root]		580		
8	dwm.exe (472)	[Root]		572		

PerfView

- ETW collection and analysis tool tailored for .NET applications (but not only)
- Can be used as a sampling profiler
- Can be used as an allocation profiler
- Can be used for heap snapshot analysis

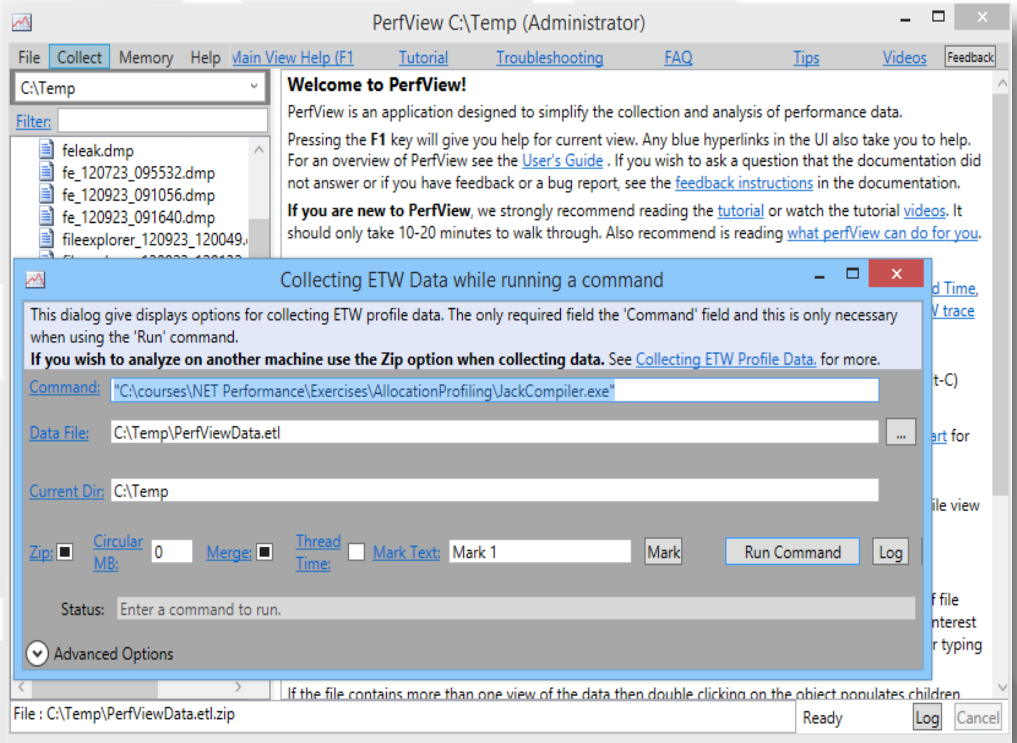
Collecting Data with PerfView

CLI

```
PerfView run app.exe
```

Option	Meaning
/MaxCollectSec:N	Stop collection after N seconds
/StartOnPerfCounter /StopOnPerfCounter	Start/stop collection based on performance counter
/Providers=… /OnlyProviders=…	Restrict to specific set of providers
/CircularMB:N	Circular logging N megabytes of newest events

GUI



Welcome to PerfView!

PerfView is an application designed to simplify the collection and analysis of performance data.

Pressing the **F1** key will give you help for current view. Any blue hyperlinks in the UI also take you to help. For an overview of PerfView see the User's Guide . If you wish to ask a question that the documentation did not answer or if you have feedback or a bug report, see the feedback instructions in the documentation.

If you are new to PerfView, we strongly recommend reading the tutorial or watch the tutorial videos. It should only take 10-20 minutes to walk through. Also recommend is reading what perfView can do for you.

If the file contains more than one view of the data then double clicking on the object populates children

This dialog give displays options for collecting ETW profile data. The only required field the 'Command' field and this is only necessary when using the 'Run' command.
If you wish to analyze on another machine use the Zip option when collecting data. See Collecting ETW Profile Data. for more.

Command: "C:\courses\NET Performance\Exercises\AllocationProfiling\JackCompiler.exe"

Data File: C:\Temp\PerfViewData.etl

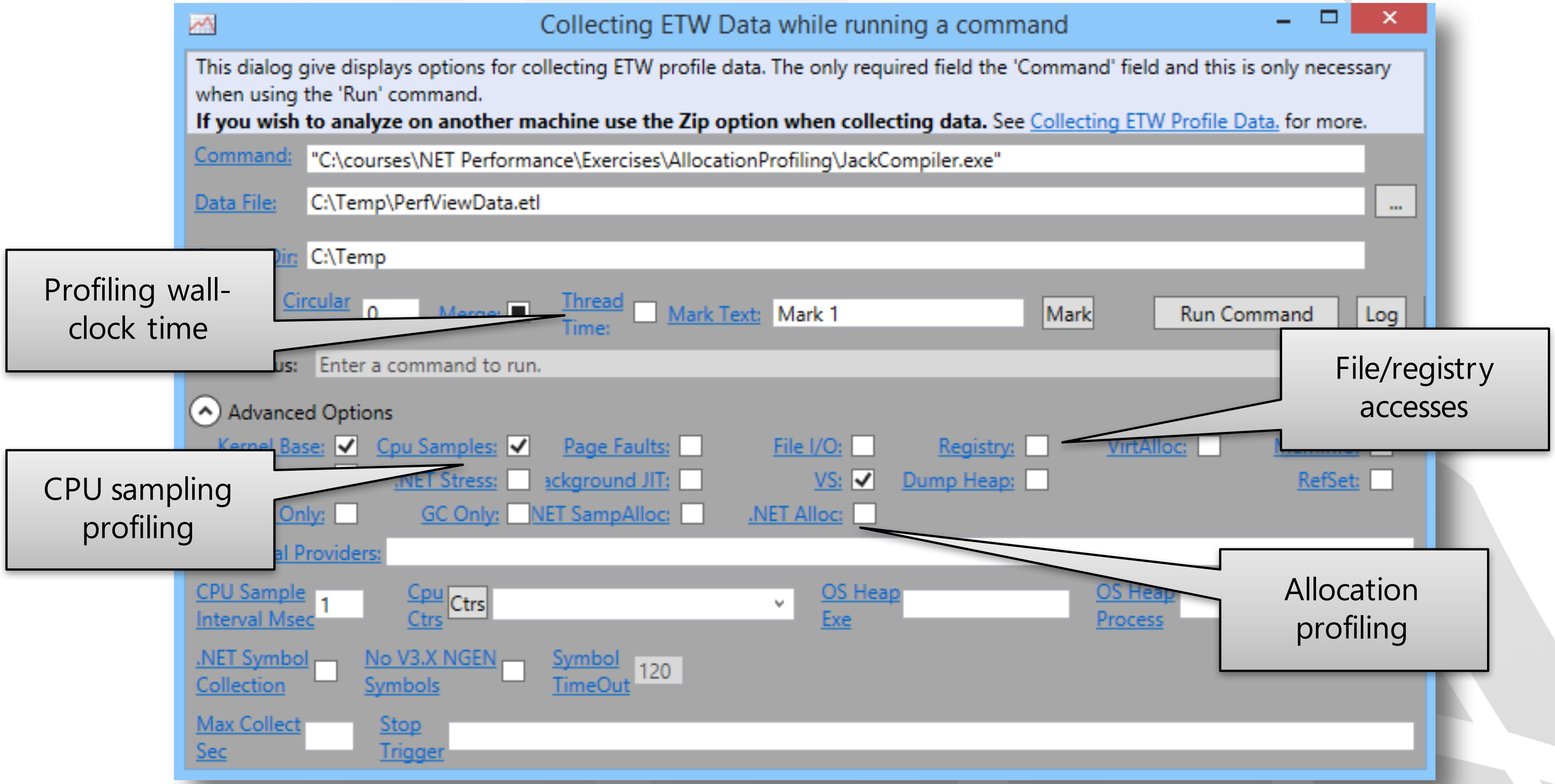
Current Dir: C:\Temp

Zip: ■ Circular MB: 0 Merge: ■ Thread Time: ☐ Mark Text: Mark 1

Status: Enter a command to run.

File : C:\Temp\PerfViewData.etl.zip

PerfView Collection Options



Collecting ETW Data while running a command

This dialog give displays options for collecting ETW profile data. The only required field the 'Command' field and this is only necessary when using the 'Run' command.
If you wish to analyze on another machine use the Zip option when collecting data. See Collecting ETW Profile Data. for more.

Command: "C:\courses\NET Performance\Exercises\AllocationProfiling\JackCompiler.exe"

Data File: C:\Temp\PerfViewData.etl

Dir: C:\Temp

Circular: 0 | Merge: ■ | Thread Time: ☐ | Mark Text: Mark 1

us: Enter a command to run.

Advanced Options

Kernel Base: ✓ | Cpu Samples: ✓ | Page Faults: ☐ | File I/O: ☐ | Registry: ☐ | VirtAlloc: ☐
.NET Stress: ☐ | ackground JIT: ☐ | VS: ✓ | Dump Heap: ☐ | RefSet: ☐
Only: ☐ | GC Only: ☐ | NET SampAlloc: ☐ | .NET Alloc: ☐

al Providers:

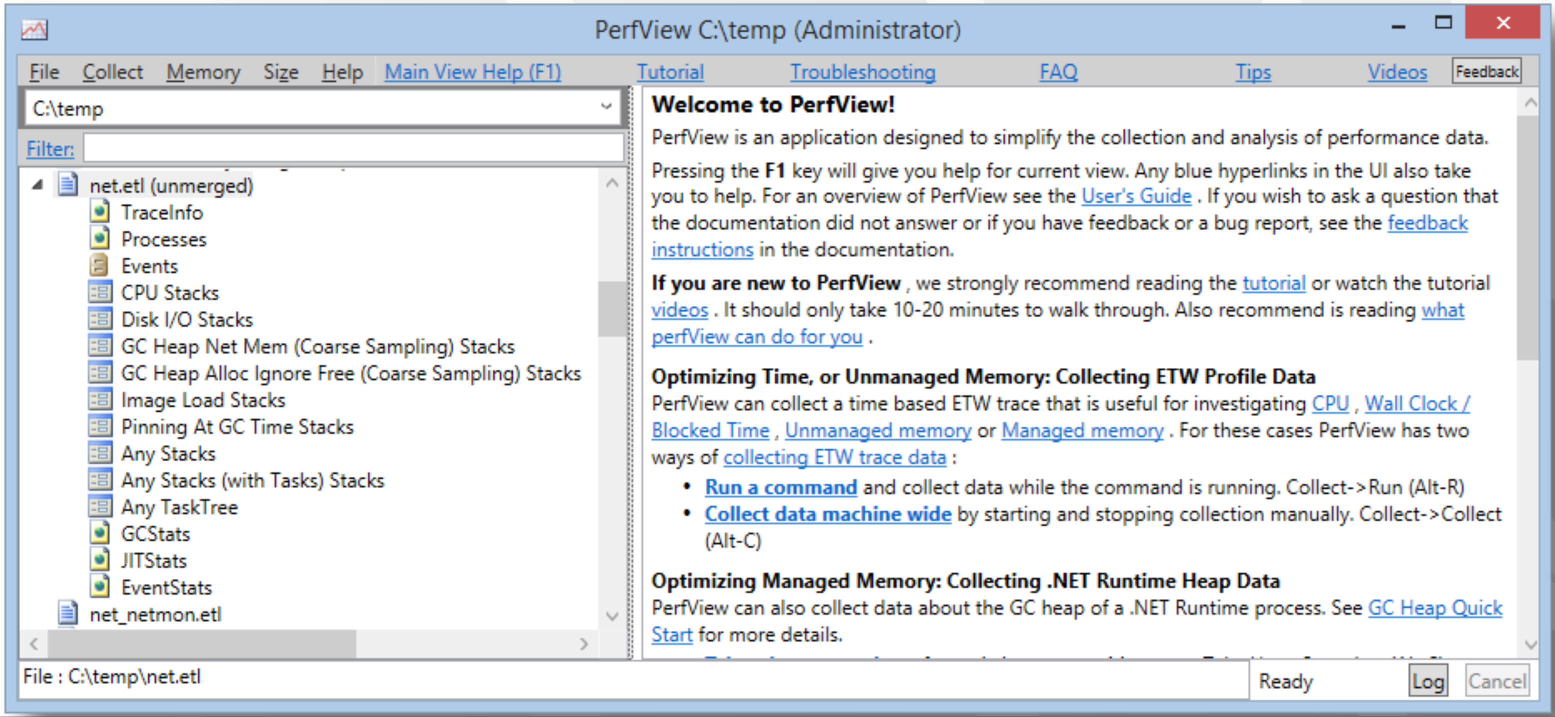
CPU Sample Interval Msec: 1 | Cpu Ctrs: | OS Heap Exe: | OS Heap Process:

.NET Symbol Collection: ☐ | No V3.X NGEN Symbols: ☐ | Symbol TimeOut: 120

Max Collect Sec: | Stop Trigger:

PerfView Reports

- PerfView has built-in support for CPU utilization, GC and JIT information, disk and file I/O, and a bunch of additional reports


PerfView C:\temp (Administrator)
File Collect Memory Size Help Main View Help (F1) Tutorial Troubleshooting FAQ Tips Videos Feedback
C:\temp
Filter:
Ready Log Cancel

net.etl (unmerged)
- TraceInfo
- Processes
- Events
- CPU Stacks
- Disk I/O Stacks
- GC Heap Net Mem (Coarse Sampling) Stacks
- GC Heap Alloc Ignore Free (Coarse Sampling) Stacks
- Image Load Stacks
- Pinning At GC Time Stacks
- Any Stacks
- Any Stacks (with Tasks) Stacks
- Any TaskTree
- GCStats
- JITStats
- EventStats

net_netmon.etl

File : C:\temp\net.etl

Welcome to PerfView!

PerfView is an application designed to simplify the collection and analysis of performance data.

Pressing the **F1** key will give you help for current view. Any blue hyperlinks in the UI also take you to help. For an overview of PerfView see the User's Guide . If you wish to ask a question that the documentation did not answer or if you have feedback or a bug report, see the feedback instructions in the documentation.

If you are new to PerfView , we strongly recommend reading the tutorial or watch the tutorial videos . It should only take 10-20 minutes to walk through. Also recommend is reading what perfView can do for you .

Optimizing Time, or Unmanaged Memory: Collecting ETW Profile Data

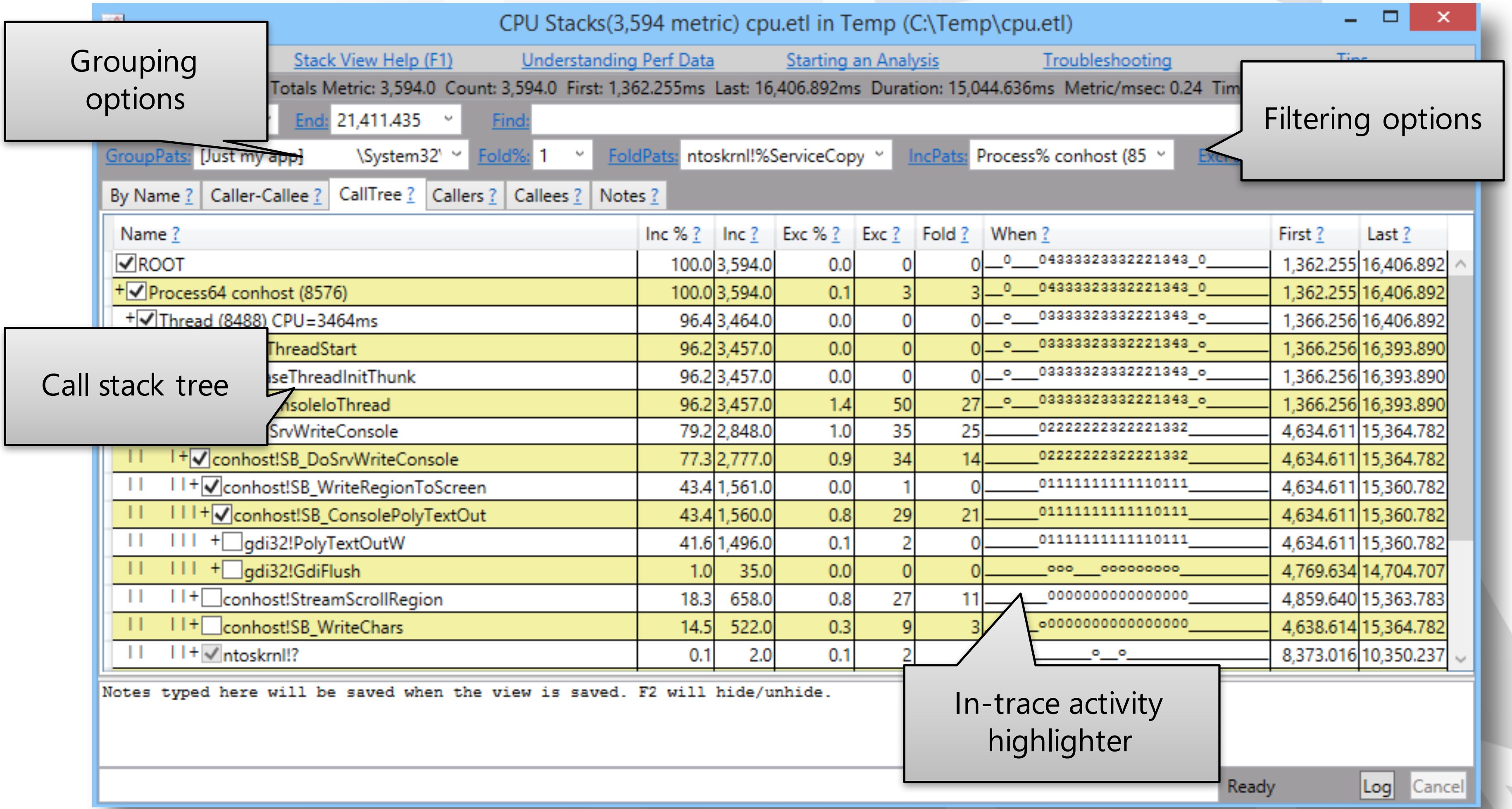
PerfView can collect a time based ETW trace that is useful for investigating CPU , Wall Clock / Blocked Time , Unmanaged memory or Managed memory . For these cases PerfView has two ways of collecting ETW trace data :

- **Run a command** and collect data while the command is running. Collect->Run (Alt-R)
- **Collect data machine wide** by starting and stopping collection manually. Collect->Collect (Alt-C)

Optimizing Managed Memory: Collecting .NET Runtime Heap Data

PerfView can also collect data about the GC heap of a .NET Runtime process. See GC Heap Quick Start for more details.

CPU Stacks



Totals Metric: 3,594.0 Count: 3,594.0 First: 1,362.255ms Last: 16,406.892ms Duration: 15,044.636ms Metric/msec: 0.24 Tim

End: 21,411.435 Find:

GroupPats: [Just my app] \System32\ Fold%: 1 FoldPats: ntoskrnl!%ServiceCopy IncPats: Process% conhost (85 Exc

Name ?	Inc % ?	Inc ?	Exc % ?	Exc ?	Fold ?	When ?	First ?	Last ?
✓ROOT	100.0	3,594.0	0.0	0	0	__0___04333323332221343_0_______	1,362.255	16,406.892
+✓Process64 conhost (8576)	100.0	3,594.0	0.1	3	3	__0___04333323332221343_0_______	1,362.255	16,406.892
+✓Thread (8488) CPU=3464ms	96.4	3,464.0	0.0	0	0	__o___03333323332221343_o_______	1,366.256	16,406.892
ThreadStart	96.2	3,457.0	0.0	0	0	__o___03333323332221343_o_______	1,366.256	16,393.890
seThreadInitThunk	96.2	3,457.0	0.0	0	0	__o___03333323332221343_o_______	1,366.256	16,393.890
nsoleIoThread	96.2	3,457.0	1.4	50	27	__o___03333323332221343_o_______	1,366.256	16,393.890
SrvWriteConsole	79.2	2,848.0	1.0	35	25	_______02222222322221332_________	4,634.611	15,364.782
| |+✓conhost!SB_DoSrvWriteConsole	77.3	2,777.0	0.9	34	14	_______02222222322221332_________	4,634.611	15,364.782
| | | |+✓conhost!SB_WriteRegionToScreen	43.4	1,561.0	0.0	1	0	_______01111111111110111_________	4,634.611	15,360.782
| | | | |+✓conhost!SB_ConsolePolyTextOut	43.4	1,560.0	0.8	29	21	_______01111111111110111_________	4,634.611	15,360.782
| | | | | +gdi32!PolyTextOutW	41.6	1,496.0	0.1	2	0	_______01111111111110111_________	4,634.611	15,360.782
| | | | | +gdi32!GdiFlush	1.0	35.0	0.0	0	0	________ooo___ooooooooo_________	4,769.634	14,704.707
| | | |+conhost!StreamScrollRegion	18.3	658.0	0.8	27	11	________0000000000000000_________	4,859.640	15,363.783
| | | |+conhost!SB_WriteChars	14.5	522.0	0.3	9	3	_______o0000000000000000_________	4,638.614	15,364.782
| | | |+✓ntoskrnl!?	0.1	2.0	0.1	2		__________o__o_________________	8,373.016	10,350.237

Notes typed here will be saved when the view is saved. F2 will hide/unhide.

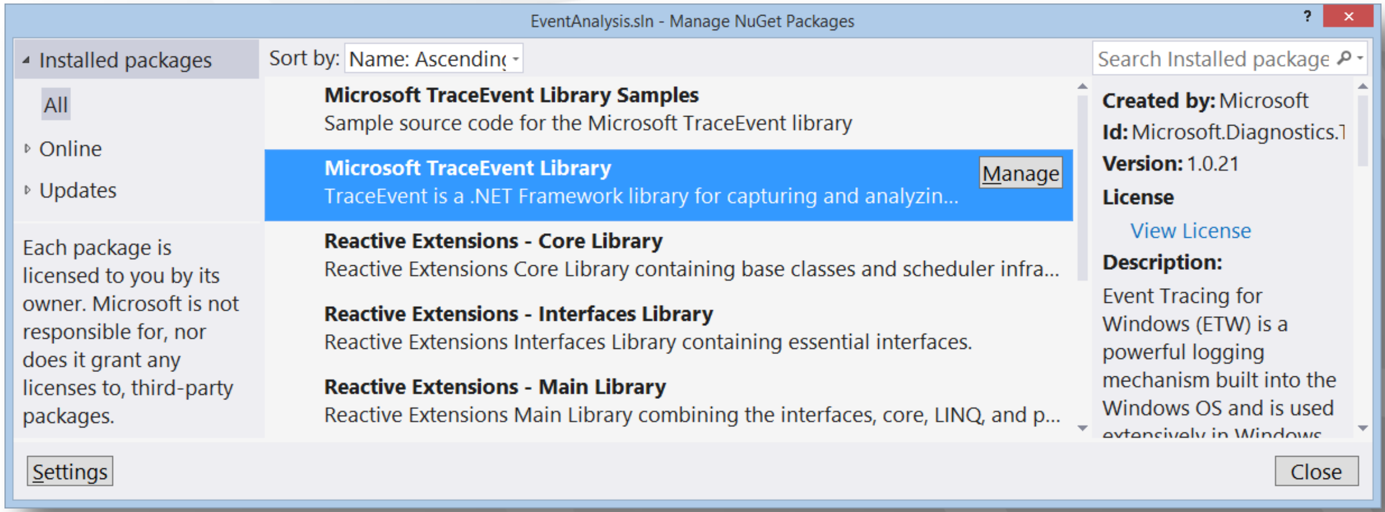
CPU Profiling with PerfView
Continuous ETW Monitoring

Lab

Programmatic ETW Analysis

Automatic ETW Analysis

- The **TraceEvent** library provides ETW analysis API
  - Understands kernel and CLR events
  - Supports call stacks (incl. managed)
  - Can start ETW sessions and/or process log files



Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Microsoft TraceEvent Library Samples
Sample source code for the Microsoft TraceEvent library

Microsoft TraceEvent Library
TraceEvent is a .NET Framework library for capturing and analyzin...

Reactive Extensions - Core Library
Reactive Extensions Core Library containing base classes and scheduler infra...

Reactive Extensions - Interfaces Library
Reactive Extensions Interfaces Library containing essential interfaces.

Reactive Extensions - Main Library
Reactive Extensions Main Library combining the interfaces, core, LINQ, and p...

Created by: Microsoft
Id: Microsoft.Diagnostics.T
Version: 1.0.21
License
View License
Description:
Event Tracing for Windows (ETW) is a powerful logging mechanism built into the Windows OS and is used extensively in Windows

Example Analysis Scenarios

- Monitor the system for CLR exceptions w/ stacks

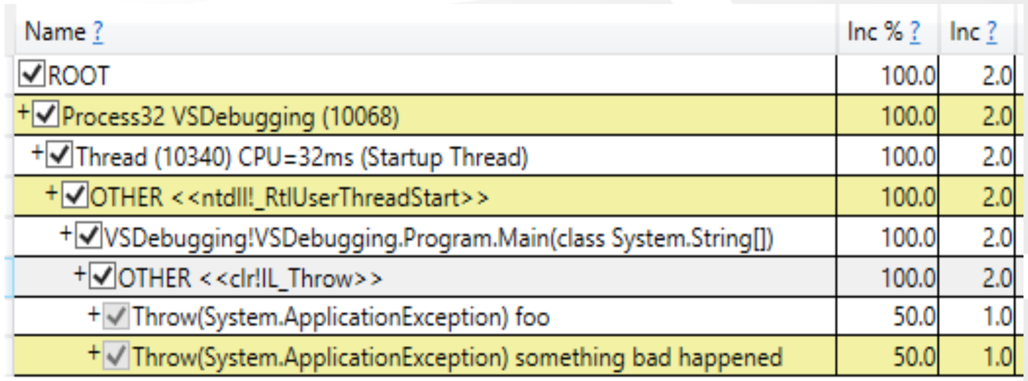
  `ExceptionTraceData`

- Get a profiling trace and look for regressions

  `TraceLog`

  `SampledProfileTraceData`

  `TraceCallStack`



Name ?	Inc % ?	Inc ?
✓ROOT	100.0	2.0
+✓Process32 VSDebugging (10068)	100.0	2.0
+✓Thread (10340) CPU=32ms (Startup Thread)	100.0	2.0
+✓OTHER <<ntdll!_RtlUserThreadStart>>	100.0	2.0
+✓VSDebugging!VSDebugging.Program.Main(class System.String[])	100.0	2.0
+✓OTHER <<clr!IL_Throw>>	100.0	2.0
+✓Throw(System.ApplicationException) foo	50.0	1.0
+✓Throw(System.ApplicationException) something bad happened	50.0	1.0

Trace Analysis Example

```
var traceLog = TraceLog.OpenOrConvert("trace.etl");
var process = traceLog.Processes.LastProcessWithName(...);
var symbolReader = new SymbolReader(Console.Out, symPath);

foreach (var exc in
   process.EventsInProcess.ByEventType<ExceptionTraceData>())
{
  Console.WriteLine(exc.ExceptionType);
  Console.WriteLine(exc.ExceptionMessage);
  var stack = exc.CallStack();
  while (stack != null)
  {
    Console.WriteLine(stack.CodeAddress.Method.FullMethodName);
    stack = stack.Caller;
  }
}
```

Trace Session Example

```
var session = new TraceEventSession("ObserveGCs");
session.EnableProvider(ClrTraceEventParser.ProviderGuid,
   TraceEventLevel.Verbose,
   (ulong)ClrTraceEventParser.Keywords.GC);

// Allocation tick every 100KB
var alloc =
   session.Source.Clr.Observe<GCAllocationTickTraceData>();
alloc.Subscribe(ad => Console.WriteLine(ad.AllocationAmount));

var gc = session.Source.Clr.Observe<GCHeapStatsTraceData>();
gc.Subscribe(cd => Console.WriteLine(cd.GenerationSize2));

session.Source.Process();
```

IntelliTrace

IntelliTrace

- IntelliTrace is a Visual Studio feature that improves developer productivity during debugging
- **“Historical Debugging”**
- Tracks events and method call information at runtime
- Records stack trace, local variables, and custom information for each event

IntelliTrace Experiences

F5 Debugging

Live debugging from Visual Studio, unit tests, and other developer experiences

Production Debugging

Collection on production systems for later analysis on a development machine

IntelliTrace Collection Modes

Low impact

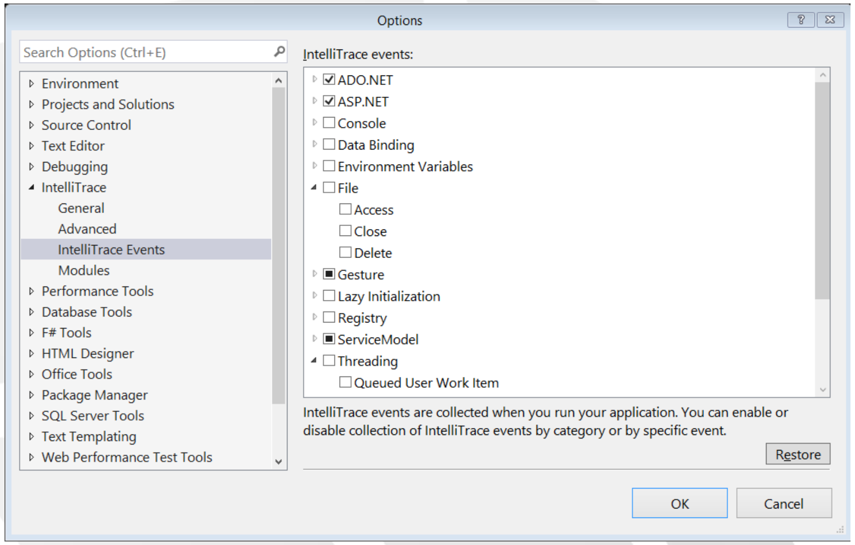
- Interesting runtime events are collected
- Low overhead if collecting low-frequency events

High impact

- Every method call is collected
- Up to 10x potential slowdown
- Configure for specific modules only to reduce impact

Events

- WCF, ADO.NET, file access, registry access, ASP.NET, and myriads of other events
- Can customize with your own events


Options

Search Options (Ctrl+E)

- Environment
- Projects and Solutions
- Source Control
- Text Editor
- Debugging
- IntelliTrace
  - General
  - Advanced
  - IntelliTrace Events
  - Modules
- Performance Tools
- Database Tools
- F# Tools
- HTML Designer
- Office Tools
- Package Manager
- SQL Server Tools
- Text Templating
- Web Performance Test Tools

IntelliTrace events:

- ☑ ADO.NET
- ☑ ASP.NET
- ☐ Console
- ☐ Data Binding
- ☐ Environment Variables
- ☐ File
  - ☐ Access
  - ☐ Close
  - ☐ Delete
- ■ Gesture
- ☐ Lazy Initialization
- ☐ Registry
- ■ ServiceModel
- ☐ Threading
  - ☐ Queued User Work Item

IntelliTrace events are collected when you run your application. You can enable or disable collection of IntelliTrace events by category or by specific event.

What Exactly Is Collected?

- Parameters and return values
- Reference type locals
  - For each referenced object, whether or not it was there, but not its contents

```
void ReadTweets(string account)
{
  var tweets = GetTweets(account);
  int count = 3;
  for (int i = 0; i < count; ++i)
    DisplayTweet(tweets[i]);
}
```

Collecting IntelliTrace Logs

- Visual Studio saves .itrace files from each run

- IntelliTrace stand-alone collector

```
IntelliTraceSC.exe launch /cp:plan.xml app.exe
```

- PowerShell cmdlets for ASP.NET/SharePoint

```
Start-IntelliTraceCollection "MyAppPool" plan.xml C:\
```

- Microsoft Test Manager
- Azure Cloud Services

Extending IntelliTrace Events

- Add your events to the collection plan XML
  - IntelliTrace can generate an event from any method in your code or framework code
  - Custom parameter formatting is available

```
<DiagnosticEventSpecification enabled="true">
  <CategoryId>gc</CategoryId>
  <SettingsName>Full collection</SettingsName>
...
    <Bindings>
      <Binding>
...
        <TypeName>System.GC</TypeName>
        <MethodName>Collect</MethodName>
        <ShortDescription>
          Garbage collection forced by the app
        </ShortDescription>
...
</DiagnosticEventSpecification>
```

Collecting IntelliTrace Logs

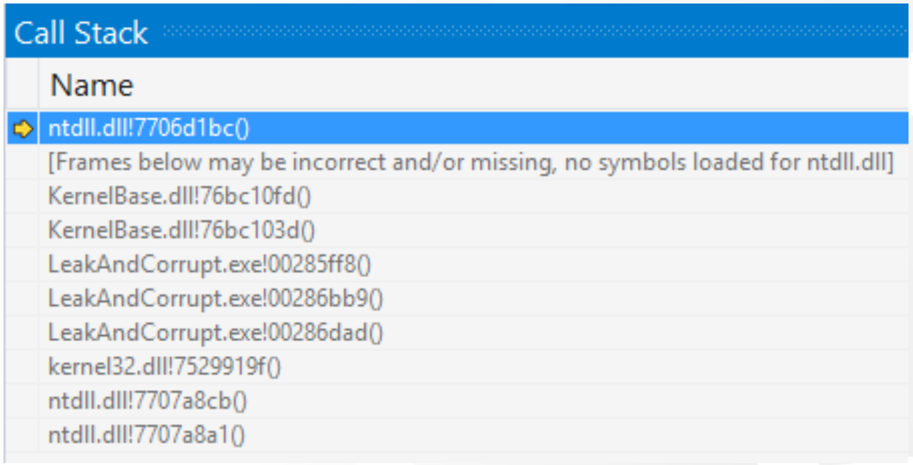
Lab

Debugging Symbols

Debugging Symbols

- *Debugging symbols* (.pdb files) link runtime memory addresses to function names, source file names and line numbers
  - Without native symbols, it's impossible to debug
  - Without managed symbols, it's harder but not impossible
- Debugging symbols make reverse engineering easier

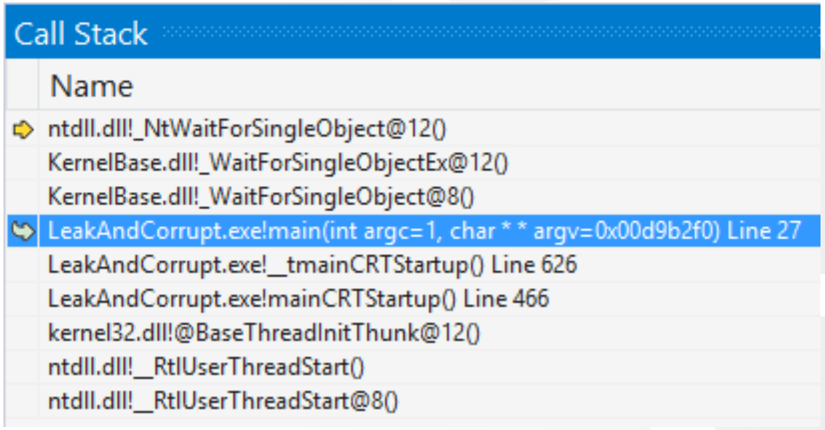
Symbols in C++



```
Call Stack
Name
ntdll.dll!7706d1bc()
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll]
KernelBase.dll!76bc10fd()
KernelBase.dll!76bc103d()
LeakAndCorrupt.exe!00285ff8()
LeakAndCorrupt.exe!00286bb9()
LeakAndCorrupt.exe!00286dad()
kernel32.dll!7529919f()
ntdll.dll!7707a8cb()
ntdll.dll!7707a8a1()
```

All useful debug information is not available without symbols:

- Function names
- Parameter types and values
- Source file and line numbers

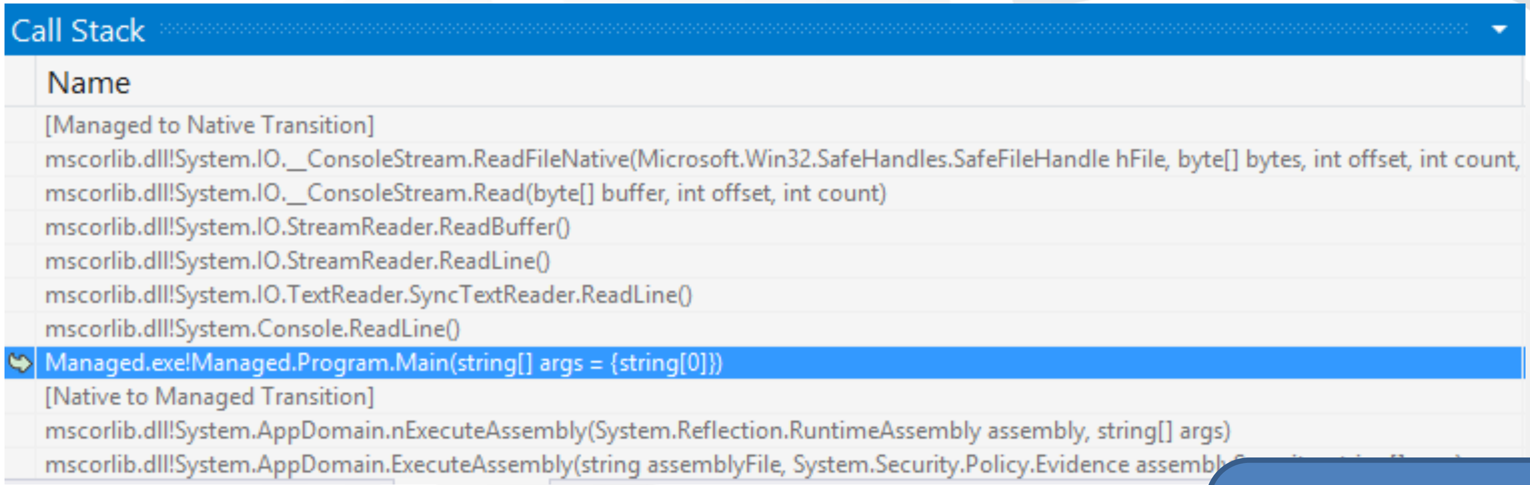


```
Call Stack
Name
ntdll.dll!_NtWaitForSingleObject@12()
KernelBase.dll!_WaitForSingleObjectEx@12()
KernelBase.dll!_WaitForSingleObject@8()
LeakAndCorrupt.exe!main(int argc=1, char * * argv=0x00d9b2f0) Line 27
LeakAndCorrupt.exe!__tmainCRTStartup() Line 626
LeakAndCorrupt.exe!mainCRTStartup() Line 466
kernel32.dll!@BaseThreadInitThunk@12()
ntdll.dll!__RtlUserThreadStart()
ntdll.dll!__RtlUserThreadStart@8()
```

Full (private) symbols include all the above information. Stripped (private) symbols do not include:

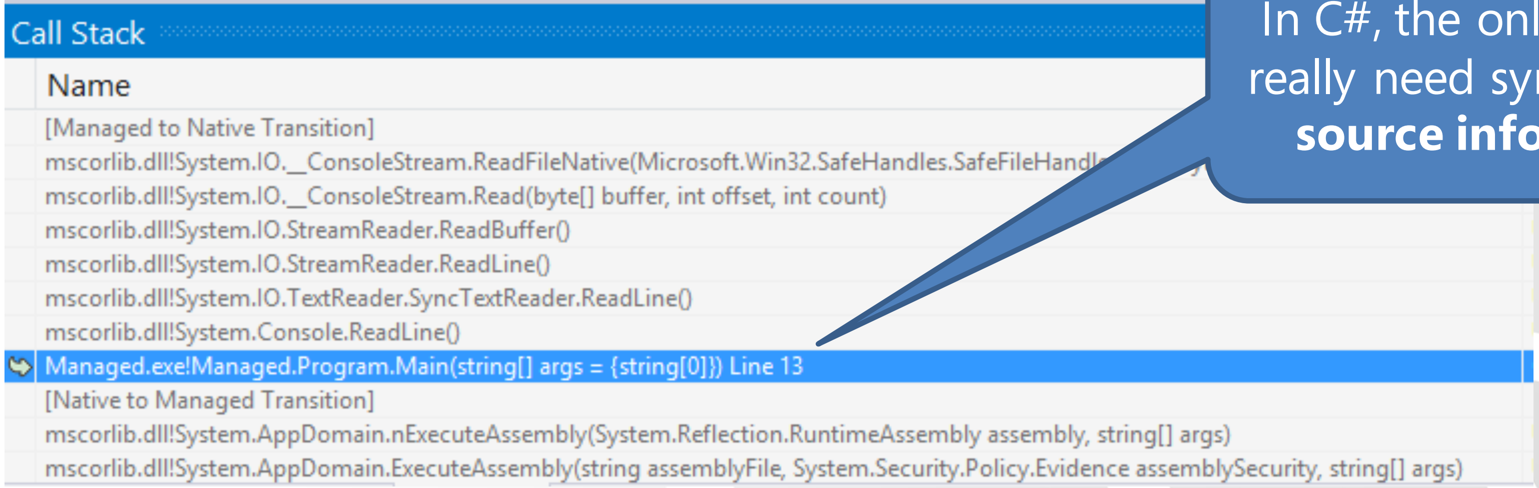
- Parameter information
- Source information

Symbols in C#



Call Stack

```
Name
[Managed to Native Transition]
mscorlib.dll!System.IO.__ConsoleStream.ReadFileNative(Microsoft.Win32.SafeHandles.SafeFileHandle hFile, byte[] bytes, int offset, int count,
mscorlib.dll!System.IO.__ConsoleStream.Read(byte[] buffer, int offset, int count)
mscorlib.dll!System.IO.StreamReader.ReadBuffer()
mscorlib.dll!System.IO.StreamReader.ReadLine()
mscorlib.dll!System.IO.TextReader.SyncTextReader.ReadLine()
mscorlib.dll!System.Console.ReadLine()
Managed.exe!Managed.Program.Main(string[] args = {string[0]})
[Native to Managed Transition]
mscorlib.dll!System.AppDomain.nExecuteAssembly(System.Reflection.RuntimeAssembly assembly, string[] args)
mscorlib.dll!System.AppDomain.ExecuteAssembly(string assemblyFile, System.Security.Policy.Evidence assembl
```



Call Stack

```
Name
[Managed to Native Transition]
mscorlib.dll!System.IO.__ConsoleStream.ReadFileNative(Microsoft.Win32.SafeHandles.SafeFileHandl
mscorlib.dll!System.IO.__ConsoleStream.Read(byte[] buffer, int offset, int count)
mscorlib.dll!System.IO.StreamReader.ReadBuffer()
mscorlib.dll!System.IO.StreamReader.ReadLine()
mscorlib.dll!System.IO.TextReader.SyncTextReader.ReadLine()
mscorlib.dll!System.Console.ReadLine()
Managed.exe!Managed.Program.Main(string[] args = {string[0]}) Line 13
[Native to Managed Transition]
mscorlib.dll!System.AppDomain.nExecuteAssembly(System.Reflection.RuntimeAssembly assembly, string[] args)
mscorlib.dll!System.AppDomain.ExecuteAssembly(string assemblyFile, System.Security.Policy.Evidence assemblySecurity, string[] args)
```

In C#, the only thing we really need symbols for is **source information**

Generating Symbols

- On by default in Debug and Release configurations
  - In C++, make sure both the compiler and the linker are configured to generate debug information
- Shipping symbols to customer machines:
  - Native code symbols make reverse engineering easier
  - Can generate stripped symbols for native code (see **PDBCopy.exe** utility or `/pdbstripped:<file>` linker switch for C++)
  - Managed symbols are not worse than a decompiler

Symbols for Microsoft Binaries

- We use Microsoft binaries all the time
  - Microsoft Visual C++ Runtime
  - MFC, ATL
  - Common Language Runtime (CLR)
  - .NET Framework classes
  - Windows itself
  - Microsoft-provided drivers
- Many of them call our code or are called by it
  - Without Microsoft symbols, some parts of your call stack might not be resolved properly

Symbols for Microsoft Binaries

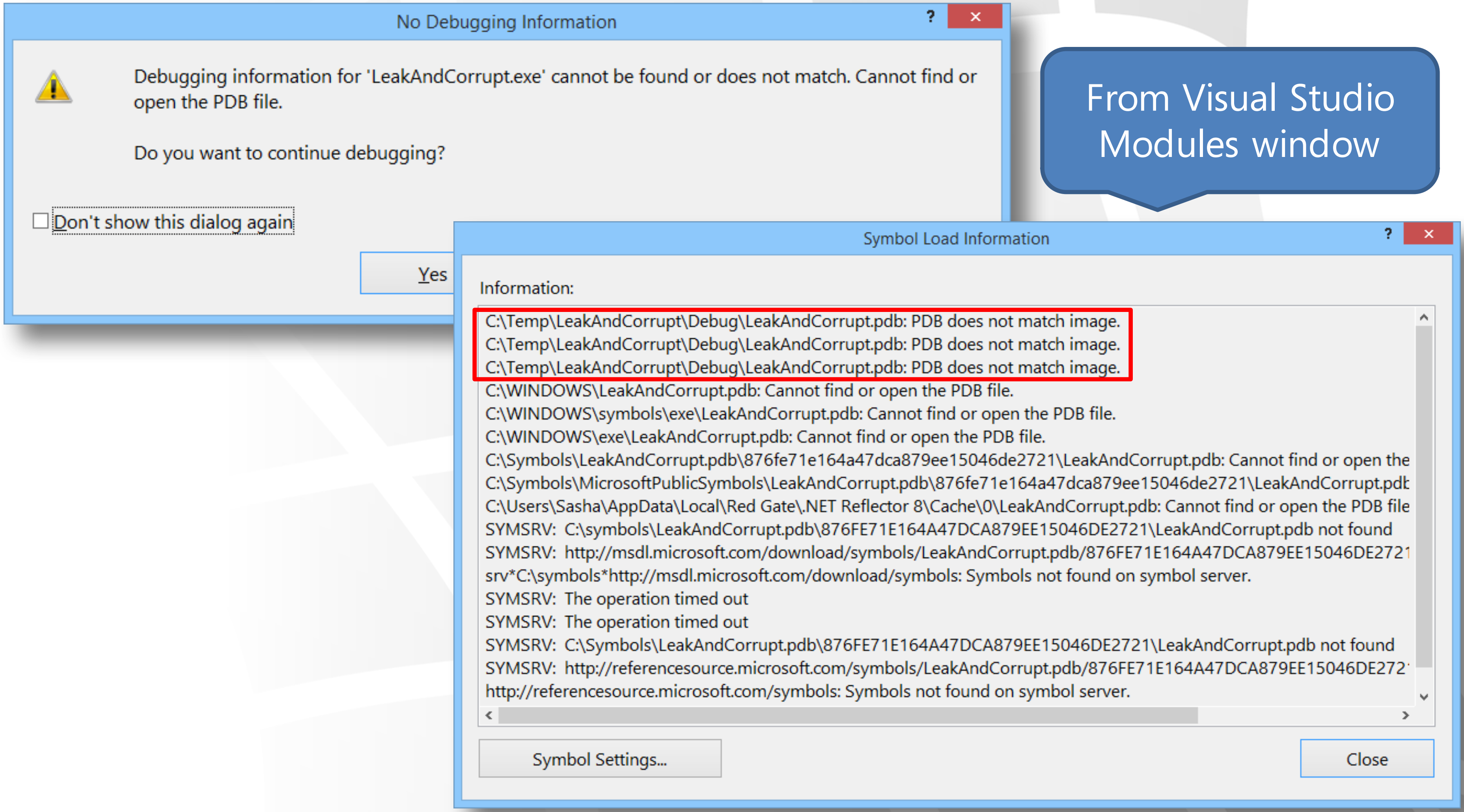
- Microsoft has a *public symbol server* with PDB files for Microsoft binaries
  - http://msdl.microsoft.com/download/symbols
- No need to download symbols manually
  - But it's possible, for offline scenarios
- Configure _NT_SYMBOL_PATH environment variable
  - And/or configure individual debuggers

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

Troubleshooting Symbol Loading

- The **symchk.exe** utility (Debugging Tools for Windows) can download specific symbols
  - Reports any missing symbols, blocked network call, and other reasons
  - Can use in offline scenarios – generate a manifest and download based on that: http://s.sashag.net/19S01wF
- In WinDbg, use `!sym noisy` and `.reload` to inspect symbol load failures
- Critical to get symbols right before starting any debugging work!

Example of Mismatched Symbols



No Debugging Information

Debugging information for 'LeakAndCorrupt.exe' cannot be found or does not match. Cannot find or open the PDB file.

Do you want to continue debugging?

☐ Don't show this dialog again

Symbol Load Information

Information:

```
C:\Temp\LeakAndCorrupt\Debug\LeakAndCorrupt.pdb: PDB does not match image.
C:\Temp\LeakAndCorrupt\Debug\LeakAndCorrupt.pdb: PDB does not match image.
C:\Temp\LeakAndCorrupt\Debug\LeakAndCorrupt.pdb: PDB does not match image.
C:\WINDOWS\LeakAndCorrupt.pdb: Cannot find or open the PDB file.
C:\WINDOWS\symbols\exe\LeakAndCorrupt.pdb: Cannot find or open the PDB file.
C:\WINDOWS\exe\LeakAndCorrupt.pdb: Cannot find or open the PDB file.
C:\Symbols\LeakAndCorrupt.pdb\876fe71e164a47dca879ee15046de2721\LeakAndCorrupt.pdb: Cannot find or open the
C:\Symbols\MicrosoftPublicSymbols\LeakAndCorrupt.pdb\876fe71e164a47dca879ee15046de2721\LeakAndCorrupt.pdb
C:\Users\Sasha\AppData\Local\Red Gate\.NET Reflector 8\Cache\0\LeakAndCorrupt.pdb: Cannot find or open the PDB file
SYMSRV:  C:\symbols\LeakAndCorrupt.pdb\876FE71E164A47DCA879EE15046DE2721\LeakAndCorrupt.pdb not found
SYMSRV:  http://msdl.microsoft.com/download/symbols/LeakAndCorrupt.pdb/876FE71E164A47DCA879EE15046DE2721
srv*C:\symbols*http://msdl.microsoft.com/download/symbols: Symbols not found on symbol server.
SYMSRV:  The operation timed out
SYMSRV:  The operation timed out
SYMSRV:  C:\Symbols\LeakAndCorrupt.pdb\876FE71E164A47DCA879EE15046DE2721\LeakAndCorrupt.pdb not found
SYMSRV:  http://referencesource.microsoft.com/symbols/LeakAndCorrupt.pdb/876FE71E164A47DCA879EE15046DE272
http://referencesource.microsoft.com/symbols: Symbols not found on symbol server.
```

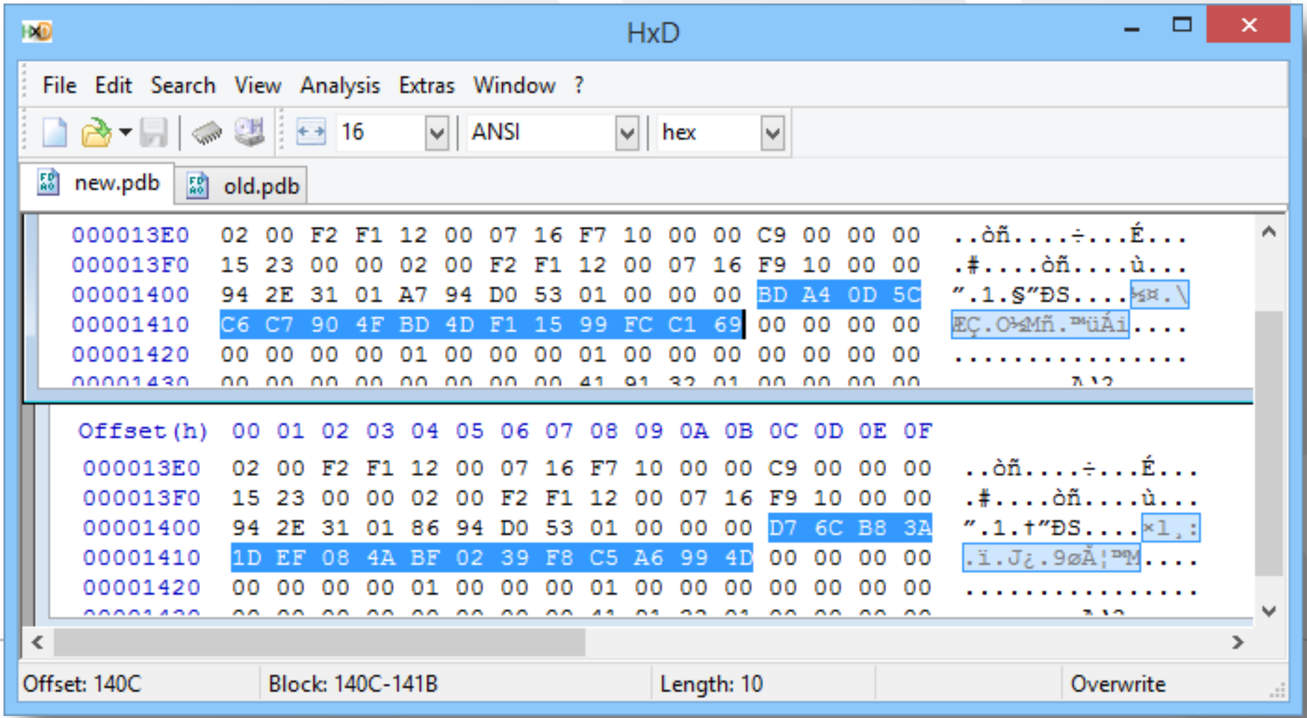
From Visual Studio Modules window

Symchk Diagnostics

```
> symchk.exe /v LeakAndCorrupt.exe /s <symbol path>
...
PdbSignature        {5C0DA4BD-C7C6-4F90-BD4D-F11599FCC169}
...
SYMCHK: LeakAndCorrupt.exe   FAILED  - LeakAndCorrupt.pdb
   mismatched or not found
...
SYMCHK: FAILED files = 1
SYMCHK: PASSED + IGNORED files = 0
```

PDB Signatures

- Even if you compile the exact same source on the exact same system, the PDB contains a unique signature that changes every time you build:



```
000013E0  02 00 F2 F1 12 00 07 16 F7 10 00 00 C9 00 00 00  ..òñ....÷...É...
000013F0  15 23 00 00 02 00 F2 F1 12 00 07 16 F9 10 00 00  .#....òñ....ù...
00001400  94 2E 31 01 A7 94 D0 53 01 00 00 00 BD A4 0D 5C  ”.1.§”ÐS....½¤.\
00001410  C6 C7 90 4F BD 4D F1 15 99 FC C1 69 00 00 00 00  ÆÇ.O½Mñ.™üÁi....
00001420  00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00  ................
00001430  00 00 00 00 00 00 00 00 41 01 32 01 00 00 00 00

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

000013E0  02 00 F2 F1 12 00 07 16 F7 10 00 00 C9 00 00 00  ..òñ....÷...É...
000013F0  15 23 00 00 02 00 F2 F1 12 00 07 16 F9 10 00 00  .#....òñ....ù...
00001400  94 2E 31 01 86 94 D0 53 01 00 00 00 D7 6C B8 3A  ”.1.†”ÐS....×l¸:
00001410  1D EF 08 4A BF 02 39 F8 C5 A6 99 4D 00 00 00 00  .ï.J¿.9øÅ¦™M....
00001420  00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00  ................
00001430  00 00 00 00 00 00 00 00 41 01 32 01 00 00 00 00
```

Offset: 140C | Block: 140C-141B | Length: 10 | Overwrite

Downloading Symbol Packages

- Windows symbols are available as a package online
  - http://msdn.microsoft.com/en-us/windows/hardware/gg463028
  - Make sure the service pack matches
  - Hotfixes might require manual patching with **symchk.exe**
- .NET Framework symbols ship separately
  - http://referencesource.microsoft.com/netframework.aspx
  - Hotfixes still problematic, CLR versions change all the time (check QFE version on PDB and DLL files)

Maintaining a Symbol Store

- It’s possible to maintain a private symbol store
- Use **symstore.exe** from Debugging Tools for Windows

```
> symstore add /r /f C:\MyApp\bin\*.pdb /s \\symsrv\syms /t
   "MyApp" /v "Build 48" /c "Manual add"

> setx _NT_SYMBOL_PATH srv*C:\Symbols*\\symsrv\syms
```

Source Servers

- Similarly to debugging symbol servers, there are also *source servers*
- Support stepping through code in the debugger
  - Even if the code is not locally available
  - Microsoft provides a source server for most of the .NET Framework assemblies
- It's also possible to set up a private source server using a set of tools shipping with the Debugging Tools for Windows

Dump Files

Dump Files

- A *user dump* is a snapshot of a running process
  - Called a user *minidump* in modern terms
- A *kernel dump* is a snapshot of the entire system
- 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

Dump File Sizes

- A dump can contain lots of information
- You can choose which data to include, and this affects what you can do with the dump later
- Example sizes for a 4GB ASP.NET process that has some unmanaged components:
  - Minidump with full memory – 4.2GB
  - Minidump with no extras – 4MB
  - Minidump with CLR heap only – 1.5GB (https://github.com/goldshtn/minidumper)
- Make sure to compress dumps before moving

Limitations of Dump Files

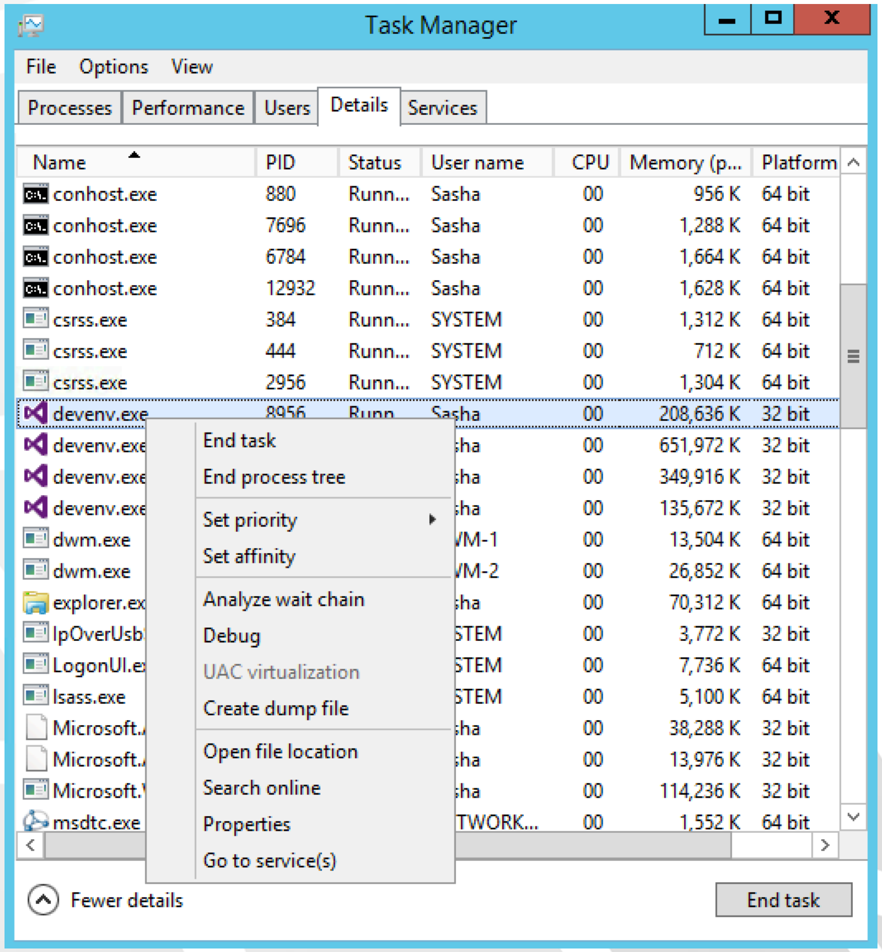
- A dump file is a static snapshot
- You can’t debug a dump, just analyze it
- Sometimes a repro is required (or more than one repro)
- Sometimes several dumps must be compared

Taxonomy of Dumps

- *Crash dumps* are dumps generated when an application crashes
  - Do not rely on a human to determine the precise moment when to capture a dump
- *Hang dumps* are dumps generated on-demand at a specific moment in time
  - Often used to diagnose hangs or infinite loops
  - Usually (but not always) require a human to trigger
- These are just names; the contents of the dump files are the same!

Windows Task Manager

- Task Manager, right-click and choose "Create Dump File"
  - Dump file goes in **%LOCALAPPDATA%\Temp**



Name	PID	Status	User name	CPU	Memory (p...	Platform
conhost.exe	880	Runn...	Sasha	00	956 K	64 bit
conhost.exe	7696	Runn...	Sasha	00	1,288 K	64 bit
conhost.exe	6784	Runn...	Sasha	00	1,664 K	64 bit
conhost.exe	12932	Runn...	Sasha	00	1,628 K	64 bit
csrss.exe	384	Runn...	SYSTEM	00	1,312 K	64 bit
csrss.exe	444	Runn...	SYSTEM	00	712 K	64 bit
csrss.exe	2956	Runn...	SYSTEM	00	1,304 K	64 bit
devenv.exe	8956	Runn...	Sasha	00	208,636 K	32 bit
devenv.exe			sha	00	651,972 K	32 bit
devenv.exe			sha	00	349,916 K	32 bit
devenv.exe			sha	00	135,672 K	32 bit
dwm.exe			VM-1	00	13,504 K	64 bit
dwm.exe			VM-2	00	26,852 K	64 bit
explorer.ex			sha	00	70,312 K	64 bit
IpOverUsb			STEM	00	3,772 K	32 bit
LogonUI.e			STEM	00	7,736 K	64 bit
lsass.exe			STEM	00	5,100 K	64 bit
Microsoft.			sha	00	38,288 K	32 bit
Microsoft.			sha	00	13,976 K	32 bit
Microsoft.\			sha	00	114,236 K	32 bit
msdtc.exe			TWORK...	00	1,552 K	64 bit

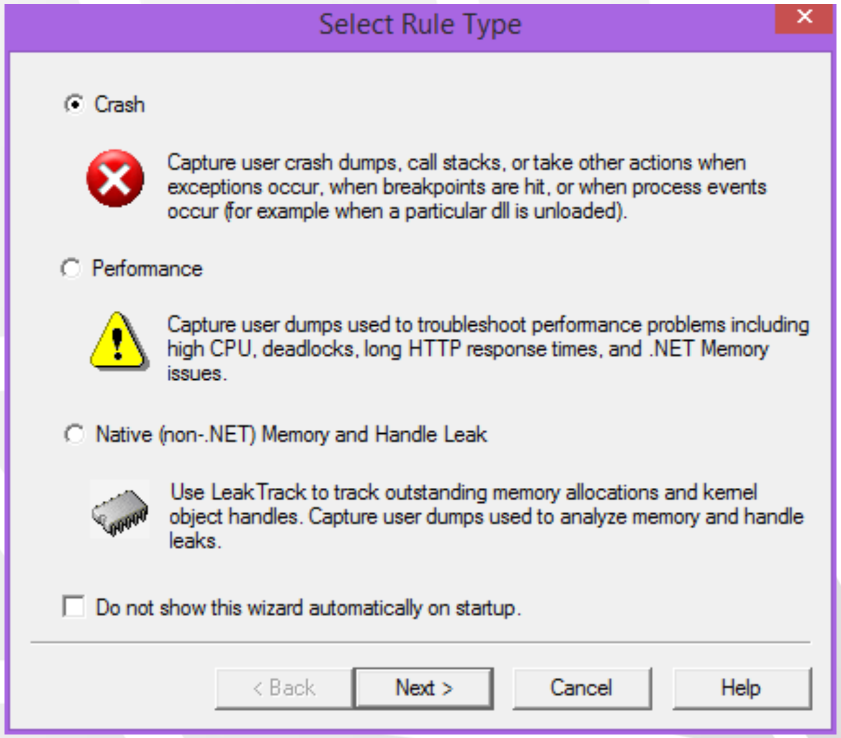
Procdump

- Sysinternals utility for creating crash / hang dumps
- Can use *process reflection* (Windows 7+) to minimize process suspension time
- Examples:

```
Procdump app.exe app.dmp
Procdump -h app.exe hang.dmp
Procdump -e app.exe crash.dmp
Procdump -c 90 app.exe excessive_cpu.dmp
Procdump -r -ma app.exe app.dmp
```

DebugDiag

- Microsoft tool for monitoring and dump generation
  - Very suitable for ASP.NET
  - Dump analysis component included


Select Rule Type

(●) Crash

Capture user crash dumps, call stacks, or take other actions when exceptions occur, when breakpoints are hit, or when process events occur (for example when a particular dll is unloaded).

( ) Performance

Capture user dumps used to troubleshoot performance problems including high CPU, deadlocks, long HTTP response times, and .NET Memory issues.

( ) Native (non-.NET) Memory and Handle Leak

Use LeakTrack to track outstanding memory allocations and kernel object handles. Capture user dumps used to analyze memory and handle leaks.

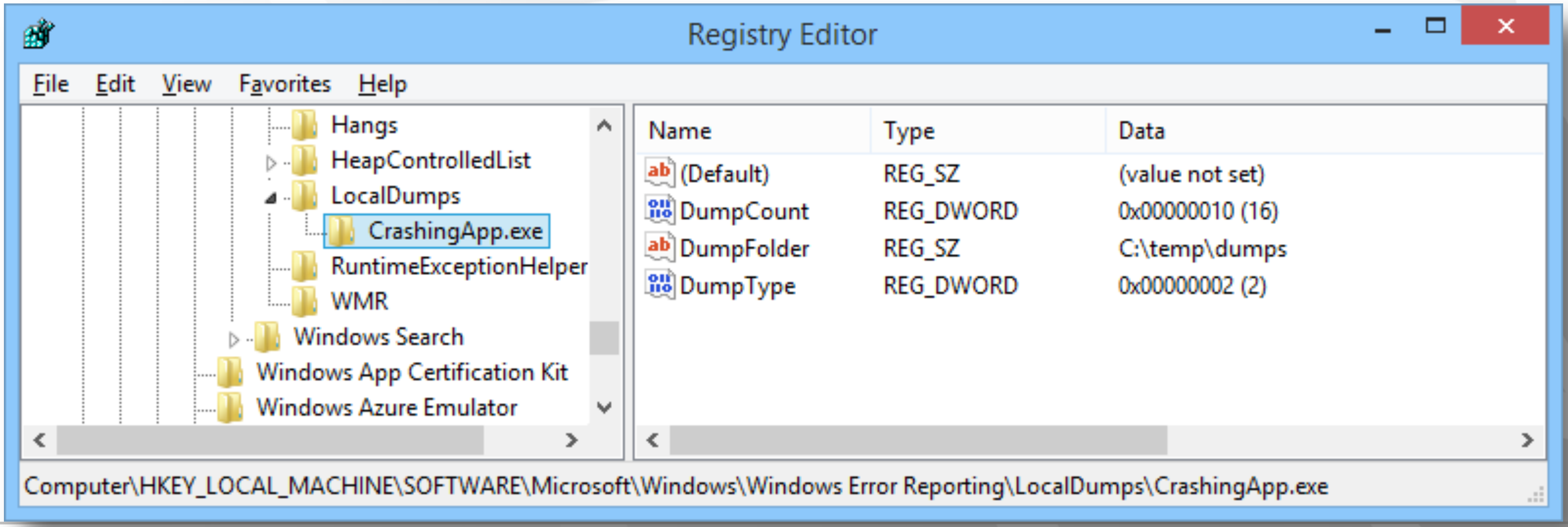
☐ Do not show this wizard automatically on startup.

Post-Mortem Debuggers

- Configured in the registry:
  - For unmanaged applications and managed as of CLR 4.0: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug**
  - For managed applications before CLR 4.0: **HKLM\SOFTWARE\Microsoft\.NETFramework**
  - Note that there are two registry keys you'd need to set on Windows x64 (the 64-bit one, and the **Wow6432Node**)
  - See http://tinyurl.com/AutoDumps

Windows Error Reporting

- WER registry key allows customization of dump file type and location
  - **LocalDumps** registry key can configure generation of local dumps (see http://tinyurl.com/localdumps)
  - Can be application-specific, not system-wide



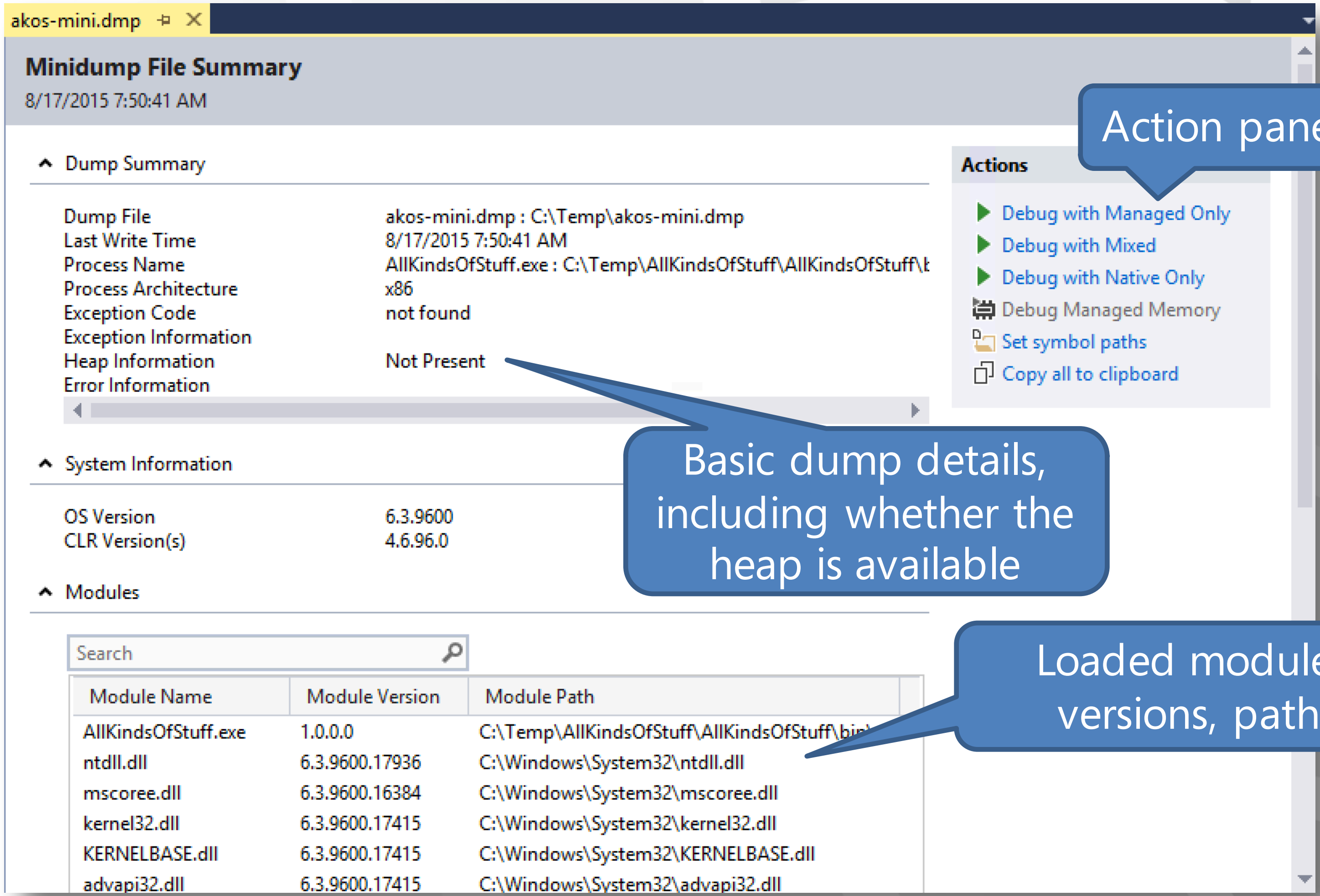
Name	Type	Data
(Default)	REG_SZ	(value not set)
DumpCount	REG_DWORD	0x00000010 (16)
DumpFolder	REG_SZ	C:\temp\dumps
DumpType	REG_DWORD	0x00000002 (2)

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\LocalDumps\CrashingApp.exe

Opening Dump Files

- Visual Studio 2010+ supports managed dump analysis
  - Requires CLR 4.0+ in the target process
  - Threads, stacks, variables, memory contents
- Visual Studio 2013+ supports managed memory analysis based on dump files
  - Object statistics, retention information (roots)

Visual Studio Dump Analysis



Minidump File Summary
8/17/2015 7:50:41 AM

Dump Summary

Dump File	akos-mini.dmp : C:\Temp\akos-mini.dmp
Last Write Time	8/17/2015 7:50:41 AM
Process Name	AllKindsOfStuff.exe : C:\Temp\AllKindsOfStuff\AllKindsOfStuff\b
Process Architecture	x86
Exception Code	not found
Exception Information	
Heap Information	Not Present
Error Information	

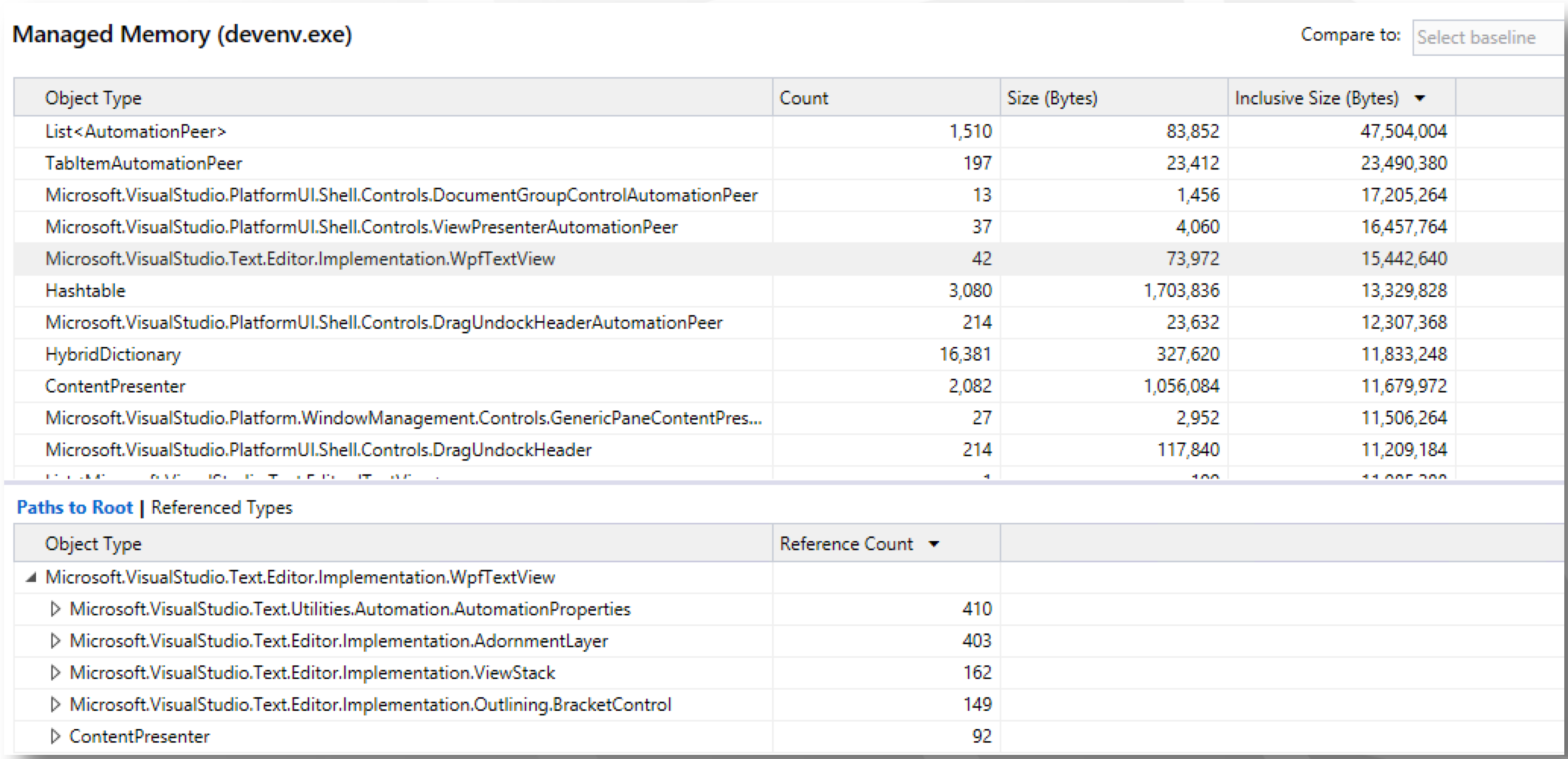
System Information

OS Version	6.3.9600
CLR Version(s)	4.6.96.0

Modules

Module Name	Module Version	Module Path
AllKindsOfStuff.exe	1.0.0.0	C:\Temp\AllKindsOfStuff\AllKindsOfStuff\bin
ntdll.dll	6.3.9600.17936	C:\Windows\System32\ntdll.dll
mscoree.dll	6.3.9600.16384	C:\Windows\System32\mscoree.dll
kernel32.dll	6.3.9600.17415	C:\Windows\System32\kernel32.dll
KERNELBASE.dll	6.3.9600.17415	C:\Windows\System32\KERNELBASE.dll
advapi32.dll	6.3.9600.17415	C:\Windows\System32\advapi32.dll

Visual Studio Memory Analysis



Managed Memory (devenv.exe)

Compare to: Select baseline

Object Type	Count	Size (Bytes)	Inclusive Size (Bytes) ▾
List<AutomationPeer>	1,510	83,852	47,504,004
TabItemAutomationPeer	197	23,412	23,490,380
Microsoft.VisualStudio.PlatformUI.Shell.Controls.DocumentGroupControlAutomationPeer	13	1,456	17,205,264
Microsoft.VisualStudio.PlatformUI.Shell.Controls.ViewPresenterAutomationPeer	37	4,060	16,457,764
Microsoft.VisualStudio.Text.Editor.Implementation.WpfTextView	42	73,972	15,442,640
Hashtable	3,080	1,703,836	13,329,828
Microsoft.VisualStudio.PlatformUI.Shell.Controls.DragUndockHeaderAutomationPeer	214	23,632	12,307,368
HybridDictionary	16,381	327,620	11,833,248
ContentPresenter	2,082	1,056,084	11,679,972
Microsoft.VisualStudio.Platform.WindowManagement.Controls.GenericPaneContentPres...	27	2,952	11,506,264
Microsoft.VisualStudio.PlatformUI.Shell.Controls.DragUndockHeader	214	117,840	11,209,184
[illegible]	[illegible]	[illegible]	[illegible]

Paths to Root | Referenced Types

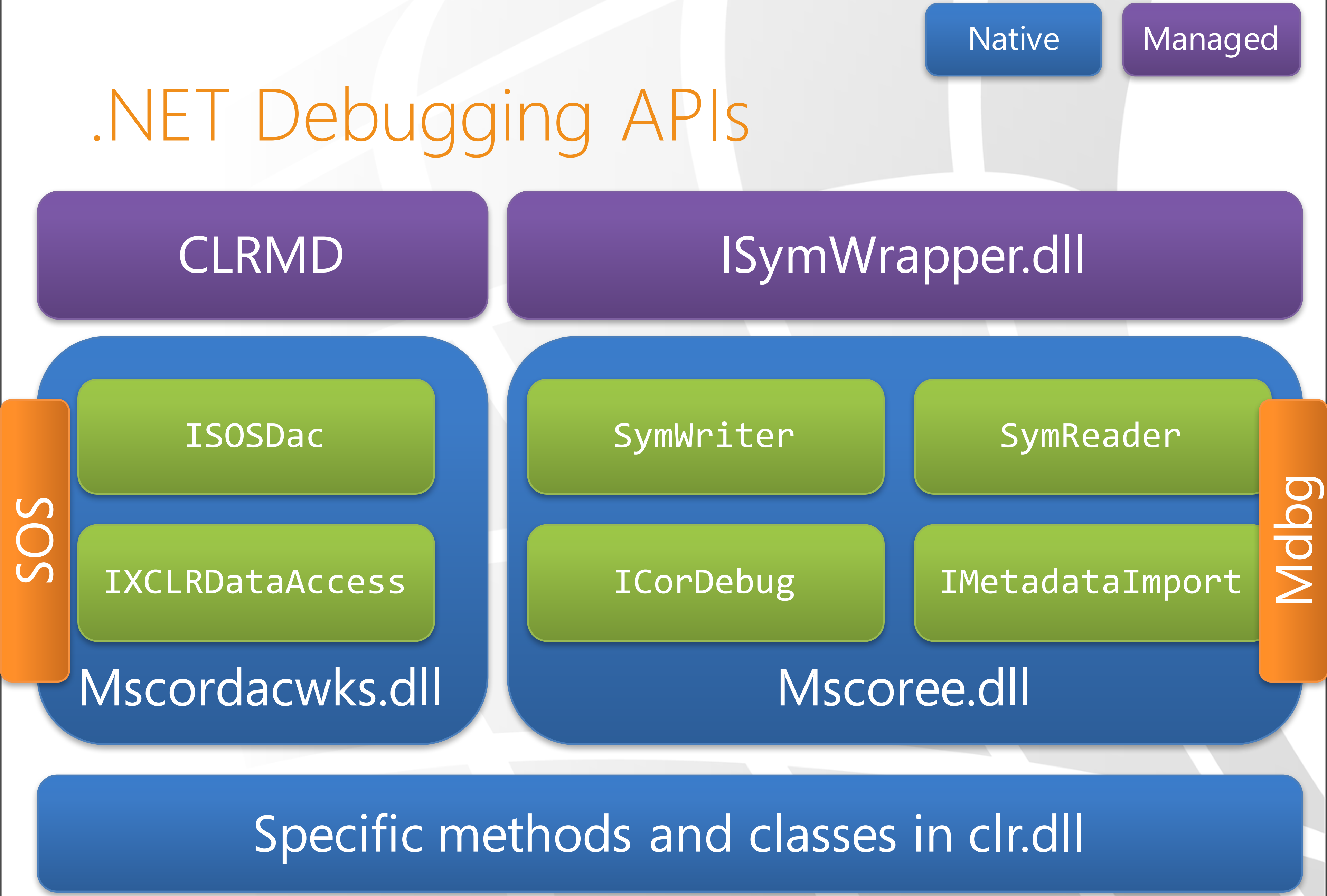
Object Type	Reference Count ▾
◢ Microsoft.VisualStudio.Text.Editor.Implementation.WpfTextView	
▷ Microsoft.VisualStudio.Text.Utilities.Automation.AutomationProperties	410
▷ Microsoft.VisualStudio.Text.Editor.Implementation.AdornmentLayer	403
▷ Microsoft.VisualStudio.Text.Editor.Implementation.ViewStack	162
▷ Microsoft.VisualStudio.Text.Editor.Implementation.Outlining.BracketControl	149
▷ ContentPresenter	92

Generating WER Dump Files
Visual Studio Dump Analysis
Visual Studio Memory Analysis

Lab

CLRMD

.NET Debugging APIs



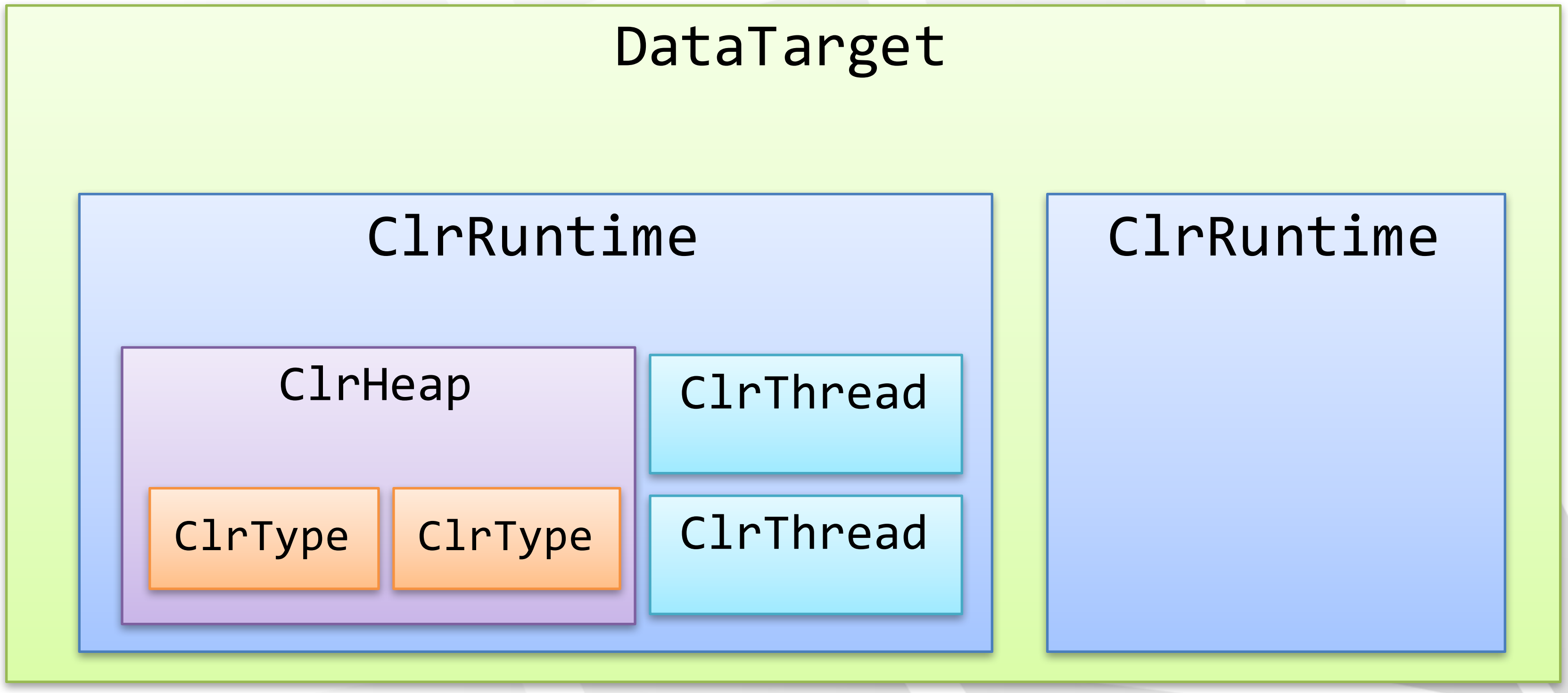
Debugging Automation Challenges

- Traditional debugging and dump analysis is done by hand
- Automation often achieved by running WinDbg commands and parsing their text output
- Debugging APIs very intricate and often undocumented (e.g. the `IXCLRDataAccess` APIs that SOS uses)

Introducing CLRMD

- **ClrMD** is a .NET library for analyzing dump files and running processes
  - Distributed through NuGet (**Microsoft.Diagnostics.Runtime** assembly)
  - Open source on GitHub
- Enables a huge variety of scenarios, including:
  - Automatic processing of many dump files
  - Continuous monitoring and inspection of production processes (threads, stacks, locks, heaps)
  - Locating specific objects and values in memory without suspending, debugging, or capturing dumps

Basic Types


DataTarget
ClrRuntime
ClrHeap
ClrType
ClrType
ClrThread
ClrThread
ClrRuntime

Connecting to a Target

- Live attach: passive, non-invasive, full
- Open dump file

```
DataTarget target = DataTarget.LoadCrashDump(@"dump.dmp");
target.AppendSymbolPath(
 "srv*C:\symbols*http://msdl.microsoft.com/download/symbols");

string dacLocation = target.ClrVersions[0].TryDownloadDac();
ClrRuntime runtime = target.CreateRuntime(dacLocation);
```

Basic Exception Analysis

```
foreach (var thread in runtime.Threads)
{
  var e = thread.CurrentException;
  if (e != null)
  {
    Console.WriteLine("Thread {0}", thread.ManagedThreadId);
    Console.WriteLine("\t{0} - {1}", e.Type.Name, e.Message);

    foreach (var frame in e.StackTrace)
      Console.WriteLine("\t" + frame.DisplayString);
  }
}
```

CLRMD Dump Analyzer
CLRMD Stack Dumper

Lab

Inspecting The Heap

- Enumerate all heap objects and statistics
- Find specific objects
- Inspect GC information (roots, finalization queues, etc.)

```
ClrHeap
    EnumerateObjects
    GetObjectType
    EnumerateRoots
ClrType
    GetSize
    EnumerateRefsOfObject
    GetFieldValue
```

Wait Information

- Threads have a list of blocking objects, which have owner threads
- Wait analysis and deadlock detection is made possible

```
ClrThread
    BlockingObjects
BlockingObject
    Reason
    Object
    HasSingleOwner
    Owner/Owners
    Waiters
```

Dynamic Heap Queries

- CLRMDExt is a library with some nice CLRMD extensions, including `ClrObject` that provides dynamic querying capabilities

```
var obj = (from o in heap.EnumerateObjects()
           let t = heap.GetObjectType(o)
           where t.Name == "MyApp.Player"
           select new ClrObject(heap, t, o, false)
          ).First();

string details = o.m_name + " " + o.m_address.m_city;
bool lastWon = o.m_games[o.m_games.m_Length - 1].m_won;
```

Running Heap Queries

Lab

Sasha Goldshtein
CTO, Sela Group

blog.sashag.net
@goldshtn

Thank You!