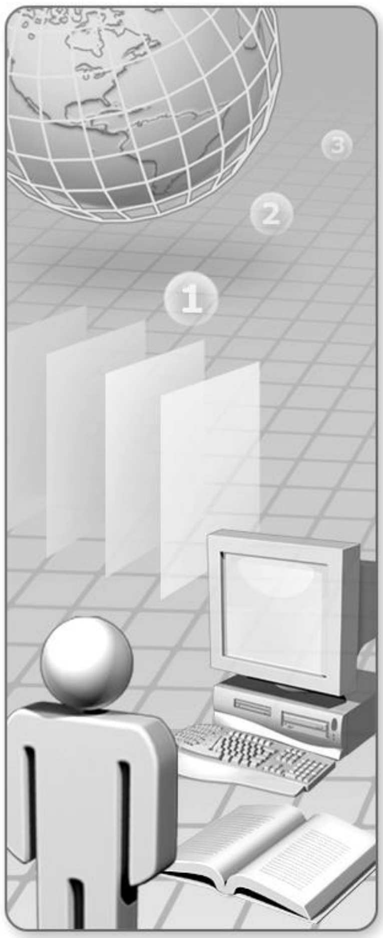
Microsoft | Services

Introduction to .NET Debugging



Lab: Introduction to .NET Debugging

Objectives

After completing this lab, you will:

- Have your computer ready and set up for the subsequent labs.
- Have attached WinDBG to a process and loaded SOS/Psscor to perform some basic debugging commands.
- Recognize and analyze the different components of a managed assembly within a debug session.
- Determine if a method has been JIT-compiled or not.
- Debug more effectively with SOS.

Estimated time to complete this lab: 115 minutes

Note: This lab focuses on the concepts in this module and as a result might not comply with Microsoft® security recommendations.

Exercise 1: Labs Setup

For this course, you will use both a Windows Forms application and a Web application and the ASP.NET LabFiles if you are planning to do the optional ASP.NET Labs. In Exercise 1, you will ensure that both are installed on your computer.

Task Description

1. Install the LabFiles:
   - A. Your instructor will provide you with **APLD.NET.msi** or the appropriate version of the installer for this course.
   - B. Run the installer. Upon completion run the C:\LabFiles\tools\Support\SetPassword.exe utility to set the password of the website you will use in later labs. Record the password that you entered here: ________________________

2. From this point on in the lab manual the installation directory will be referred to as <FotoVisionDir>.
   - A. <FotoVisionDir> = C:\LabFiles\FotoVision\4.0
3. Rebuild the solutions in Debug
   - A. Go to <FotoVisionDir>\C#\Desktop\Source and double click on the Solution (*.SLN) file to launch it in Visual Studio and build the Debug version
   - B. Go to <FotoVisionDir>\VB\LoginManager\Source and double click on the Solution (*.SLN) file to launch it in Visual Studio and build the Debug version.
   - C. Go to <FotoVisionDir>\VB\Web\Source and double click on the Solution (*.SLN) file to launch it in Visual Studio and build the Debug version
4. You should not get any build errors, and you can continue on to the next exercise.

Exercise 2: Using WinDBG with SOS

In this exercise, you will use WinDBG with SOS to debug an application.

Note: For each and every CLR version there is a different version of managed debug extension called SOS. For v2.0 Microsoft made public it's internally used version of SOS in 2010 called Psscor2. This is a superset of the v2 SOS thus it is recommended using Psscor2 when debugging 2.0 applications. Similarly for .NET 4.0 however there is **no** version for .NET 4.5 available.
The extensions SosEx and NetExt have similar capabilities.

Note: All labs in this course assume you have Debugging Tools for Windows installed in the "C:\Debuggers" directory for convenience of the short path working in cmd prompt.

The Debugging Tools for Windows is available in the latest Windows SDK. There is no standalone install available.

Debugging Extensions:

SosEx.dll is available at http://www.stevestechspot.com/

NetExt.dll is available at http://netext.codeplex.com/

Psscor2.dll **(.NET 2.x - .NET 3.5 only)** is available at:
http://www.microsoft.com/downloads/details.aspx?FamilyID=5c068e9f-ebfe-48a5-8b2f-0ad6ab454ad4&displaylang=en

Psscor4.dll **(.NET 4.0 only)** is available at:
http://www.microsoft.com/downloads/en/details.aspx?FamilyID=a06a0fea-a4d4-434e-a527-d6afa2e552dd

you should use x86 version for 32-bit process and AMD64 version for 64-bit process

Debugging Tools:

PerfView.exe is available at http://www.microsoft.com/en-us/download/details.aspx?id=28567

Procdump.exe is available at https://technet.microsoft.com/de-de/sysinternals/dd996900 and is part of the Microsoft Sysinternals Suite at https://technet.microsoft.com/de-de/sysinternals

A command tree (CommandTreePublic.txt) for WinDBG with the most popular commands can be found nearby the Lab Manual- To use the command tree

1. copy it into the directory of your debugger (c:\debuggers)
2. To load the command tree use

.cmdtree CommandTreePublic.txt

You can automatically load the command tree using
windbg.exe -c ".cmdtree CommandTreePublic.txt"
within a shortcut.

Task Description

1. Configure your settings to make Debugging Tools for Windows easier to access and use; edit your PATH environment variable

   A. On the Start menu, right-click **Computer** and select **Properties.**

   B. On the left pane click on **Advanced system settings.**

   C. Click **Environment Variables**.

   D. In the System Variables area, scroll down and find the Path variable. Select it, and then click **Edit**.

   E. At the end of the existing Variable Value, type “;**C:\Debuggers**”. If you have Debugging Tools for Windows installed in a folder other than the C:\Debuggers folder, type that folder name instead.

   F. Click **OK** two times and then close the System Properties window.

2. Set your symbol path.

   A. On the Start menu, click Run, and then type “**WinDBG**”.

   B. On the File menu, click **Symbol File Path**.

   C. Type the following:

   <FotoVisionDir>\C#\Desktop\Bin;srv*c:\symsrv*http://msdl.microsoft.com/download/symbols

   D. Click **OK**.

   E. On the File menu, click **Save Workspace**.

   F. On the File menu, click **Exit**.

3. Attach WinDBG to the sample application.

   A. Navigate to <FotoVisionDir>\C#\Desktop\bin and run FotoVision.exe.

B. On the Start menu, click Run, and then type “**WinDBG**”.

C. On the File menu, click **Attach to a Process…**

D. On the resulting list, click **FotoVision.exe** and then click **OK.**

E. On the Workspace 'base' window, check the checkbox "Don't ask again in this Windbg session" and click Yes if a dialog pops up.

4. Investigate the threads by using native commands.

A. Type **“~*kb**” and hit Enter to see the stacks for all the threads.

B. Note that thread 0’s stack appears to be broken, or to have symbol problems.

```
0:000> ~*kb

.  0  Id: 3eb4.3ae0 Suspend: 1 Teb: 7e71b000 Unfrozen
ChildEBP RetAddr  Args to Child
008ff254 67f84cb8 814bb20f 74a08bc8 008ff558 USER32!NtUserWaitMessage+0xc
008ff2d8 67f84749 00000000 ffffffff 00000000 System_Windows_Forms_ni+0x1b4cb8
008ff32c 67f845c2 025d3c14 00000000 00000000 System_Windows_Forms_ni+0x1b4749
008ff358 67f66781 025d3c14 008ff47c 00000000 System_Windows_Forms_ni+0x1b45c2
008ff3d8 74a02652 00ae58b0 008ff438 74a11580 System_Windows_Forms_ni+0x196781
008ff3e4 74a11580 008ff47c 008ff428 74ba619f clr!CallDescrWorkerInternal+0x34
008ff438 74a1e670 00000000 00000001 008ff498 clr!CallDescrWorkerWithHandler+0x6b
008ff4b0 74b72955 008ff5ac 8444aa57 00a26d9c
clr!MethodDescCallSite::CallTargetWorker+0x152
008ff5d4 74b72891 00000000 00000001 8444aa6b clr!RunMain+0x1aa
008ff848 74b2ec8b 00000000 8444a7db 00720000 clr!Assembly::ExecuteMainMethod+0x124
008ffd48 74b2ed2e 8444a223 00000000 00000000
clr!SystemDomain::ExecuteMainMethod+0x63c
008ffda0 74b2ee42 8444a263 00000000 00000000 clr!ExecuteEXE+0x4c
008ffde0 74b32370 8444a19f 00000000 00000000 clr!_CorExeMainInternal+0xdc
008ffe1c 750aec94 84577b73 75897b50 750a0000 clr!_CorExeMain+0x4d
008ffe58 7512bbcc 7512bb40 7512bb40 008ffe7c mscoreei!_CorExeMain+0x10a
008ffe68 75897c04 7e71f000 75897be0 84dee78a MSCOREE!_CorExeMain_Exported+0x8c
008ffe7c 77cbb90f 7e71f000 8690a9fb 00000000 KERNEL32!BaseThreadInitThunk+0x24
008ffec4 77cbb8da ffffffff 77ca0700 00000000 ntdll!__RtlUserThreadStart+0x2f
008ffed4 00000000 7512bb40 7e71f000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

Why are you not able to see any recognizable function names from the application source code? Hint: It’s not a symbol issue. (If you want to refer to the application source code, you can find it in the <FotoVisionDir>\C#\Desktop\Source folder.)

Note: The exact output you will see will differ from the preceding

5. Load SOS and begin working with it:

A. **.NET 4.0** - type: "**.loadby sos clr** "

For NET 2.0 you would type: "**.loadby sos mscorwks**"

Note: From this point on the labs will simply instruct you to load SOS. Use the appropriate command from above based on the version of the Microsoft .NET Framework you are using.

B. Type "**~*e!clrstack**" to view the managed stacks of all threads.

   Which thread has a managed stack? Which threads are managed threads?

C. Type "**!threads**".

   Does your answer to the previous question agree with the output of this command?

D. To see both the managed and unmanaged stacks on thread 0—including some calls that have been popped off the stack—first make thread 0 the active thread. Type "**~0s**".

```
0:004> ~0s
eax=00000002 ebx=01954a48 ecx=0000077c edx=00000102 esi=01a602c8 edi=01a7bcf4
eip=772964f4 esp=002eeb8c ebp=002eec20 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!KiFastSystemCallRet:
772964f4 c3              ret
```

E. To view the entire stack, type "**!dumpstack**". Can you tell where the unmanaged code calls into managed code and vice versa? Try "**!dumpstack -ee**" - This will only show the managed calls in the !dumpstack output.

Note:

!DumpStack..
shows a combination of managed and native callstack but it is a raw stack, meaning that it will display any addresses on the stack pointing to code. This means that !dumpstack will **not** give a completely valid stack trace
(whereas !clrstack and kb will show only proven parts of the callstack - if functionA calls functionB we will see functionA directly below functionB in the stack)

Please see **!SosEx.mk** as well.

F. Type "**!clrstack**" to help answer the previous question.

6. Run SOS commands that relate to the managed heap.

   A. Type "**!eeheap -gc**" to see how much memory is taken up by the managed heap. *How much is taken up in the large object heap?*

B. Type "**!dumpheap -stat**" to see what kinds of objects (and how many of them) are currently alive in the heap.
How many instances of FotoVisionDesktop.PaneCaption objects currently exist? How much memory do they consume?

7. Run SOS commands that relate to objects.

A. Type "**!dso**". This is short for !DumpStackObjects.
What are the types that are in use on the current thread?

B. Copy the object address of the **System.Windows.Forms.Application+ThreadContext** type

C. Type "**!do**" and then paste the object address to learn more about the types that make up this object. (This is short for !DumpObject) You can use "!do" for any managed object's valid address but it will not work on value types.

8. End the debug session.

A. Shut down WinDBG. This will also shut down FotoVision and end the debug session.

B. If a dialog pops up asking if you want to save the workspace, select "No".

Exercise 3: Investigating Assemblies and Their Components

In this exercise, you will use SOS to investigate the different components that make up an assembly.

Task Description

0.

1. Set up your debug session:

   Open WinDBG.

   On the File menu, click **Open Executable**. Click **FotoVision.exe** from the <FotoVisionDir>\C#\Desktop\Bin directory and then click **Open**.
   If a dialog pops up, mark the checkbox and select "Yes".

The debugger should be stopped at a debug break (int 3) as shown:

```
CommandLine: C:\LabFiles\FotoVision\4.0\C#\Desktop\Bin\FotoVision.exe
Symbol search path is:
C:\LabFiles\FotoVision\4.0\C#\Desktop\Bin;srv*c:\symsrv*http://msdl.microsoft.com/
download/symbols
Executable search path is:
ModLoad: 00530000 0058e000   FotoVision.exe
ModLoad: 77c60000 77dce000   ntdll.dll
ModLoad: 75120000 75176000   C:\WINDOWS\SysWOW64\MSCOREE.DLL
ModLoad: 75880000 759c0000   C:\WINDOWS\SysWOW64\KERNEL32.dll
ModLoad: 76160000 76237000   C:\WINDOWS\SysWOW64\KERNELBASE.dll
ModLoad: 6c430000 6c4d0000   C:\WINDOWS\SysWOW64\apphelp.dll
(14b4.134c): Break instruction exception - code 80000003 (first chance)
(5f1c.3a40): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=984e0000 edx=00000000 esi=7ec13000 edi=00000000
eip=77d1415d esp=0070f5bc ebp=0070f5e8 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!LdrpDoDebuggerBreak+0x2b:
77d1415d cc              int     3
```

2. Type “**g**” in the command line of the debugger to allow the executable file to load and initialize.

3. FotoVision should be open and ready for use. On the Help menu, click **About Fotovision**.

4. Now, let's Locate FotoVision.exe:

5. In WinDBG, press **Ctrl+Break** to break back into the process.

6. Load SOS.

7. Find the address of the AppDomain that contains the binary (that is, FotoVision.exe). If you’re not sure of the SOS command that does this, type “!help”.

(Hint: We need to dump the domain. Find a suitable command in !help)

8. Once you find the AppDomain that contains the binary, copy the Assembly address of the binary.

Now, Get details for a single-module assembly:

9. Type "**!DumpAssembly <Assembly address>**". Copy the Module address of the binary in this assembly for later usage.
10. *Given the definition of "Assembly" that was supplied in this chapter, what part of the output of the "!DumpAssembly" command in this case verifies that this is a single-module assembly and not a multi-module assembly?*
11. Type "**!DumpModule <Module address>"**.
Does the resulting Assembly address match the Assembly address from the previous step, in which you dumped out the AppDomains?
What is the size of this module's metadata?

SOS features several different ways to find a managed class within a debug session. Each way has its pros and cons. The next three steps of this lab will detail three separate ways to do this, using the class that represents the Help, About form as an example.

<u>Finding the Help, About form, Method 1:</u>

12. Type "**!DumpHeap -stat**" and search the list for a class that looks like it might represent the Help, About form.

```
0:008> !dumpheap -stat
Statistics:
      MT    Count    TotalSize Class Name
73c98ca8        1           12
System.Collections.Generic.ObjectEqualityComparer`1[[System.Object, mscorlib]]
73c98774        1           12 System.Reflection.__Filters
73c9851c        1           12
System.Collections.Generic.ObjectEqualityComparer`1[[System.RuntimeType,
mscorlib]]
73c94cf8        1           12
System.Collections.Generic.GenericEqualityComparer`1[[System.String, mscorlib]]
73c91e7c        1           12 System.Reflection.AssemblyProductAttribute
73c91c24        1           12
System.Security.AllowPartiallyTrustedCallersAttribute
73c91ba0        1           12 System.CLSCompliantAttribute
73c91a20        1           12 System.Diagnostics.DebuggableAttribute
73c91198        1           12 System.Security.HostSecurityManager
73c90c28        1           12
System.Collections.Generic.ObjectEqualityComparer`1[[System.Type, mscorlib]]
73c8f448        1           12
System.Collections.Generic.ObjectEqualityComparer`1[[System.Runtime.Serialization.
MemberHolder, mscorlib]]
73c8ebc0        1           12 System.Nullable`1[[System.Boolean, mscorlib]]
```

```
73c8df38        1           12 System.Threading.ApartmentState
73c8d0e4        1           12 System.Resources.FastResourceComparer
73c8616c        1           12 System.FlagsAttribute
extra output deleted for sake of brevity
…
73c8460c      179         6444 System.Security.Permissions.FileIOPermission
72596c60      116         6496 System.Configuration.FactoryRecord
73c94b04      247         6916 System.Text.StringBuilder
67ff54ac      645         7740 System.Windows.Forms.Keys
67feb6fc      509         8144 System.Windows.Forms.LayoutEventArgs
73c55a34      141         8460 Microsoft.Win32.Win32Native+WIN32_FIND_DATA
73c95ddc      299         9568 System.IO.PathHelper
73c9626c      412         9888 System.Collections.ArrayList
73c93acc      119        11724 System.Int32[]
73c91238       69        11784 System.Collections.Hashtable+bucket[]
73c98618      463        11880 System.Reflection.CustomAttributeRecord[]
67fe9c9c      215        11900 System.Windows.Forms.PropertyStore+ObjectEntry[]
73c8bed4      388        23280 System.Reflection.RuntimeMethodInfo
73c92b74      483        77950 System.Char[]
73c55738     1894       111724 System.Object[]
73c95670      253       148045 System.Byte[]
73c921b4     5324       293384 System.String
Total 19247 objects
```

One benefit of this approach is that you don't have to know the specific syntax for the class that you are using. You should consider using this command to find a class when you are not familiar with the source code.

One drawback of this approach is that, depending on the size of the managed heap and the number of heaps, the **!dumpheap -stat** command can result in a very long list that may be tedious to traverse. Another problem is that this method relies on an object being currently in the GC heap.

Were you able to find the class that represents the Help, About form?

Finding the Help, About form, Method 2:

13. To find the MT of the class representing the Help, About form, type:

.shell -ci "!dumpheap -stat" find "Help"

Note: The .shell command execute command line commands. You can get further information in the Debugger Help File.

This is a reasonable guess, as it makes the assumption that that the name of the class may have the word "help" in it and that "form" may be too common.

Can you find the class in the output?

```
0:004> .shell -ci "!dumpheap -stat" find "Help"
.shell -ci "!dumpheap -stat" find "Help"
73c83748        1          12
System.Collections.Generic.GenericArraySortHelper`1[[System.UInt64, mscorlib]]
73c55844        1          20 System.Threading.ThreadHelper
73cae6d0        2          56 System.Threading.Mutex+MutexTryCodeHelper
00b23c8c        1          76 FotoVisionDesktop.CropHelper
73c56010        3          96
System.Runtime.CompilerServices.RuntimeHelpers+CleanupCode
73c55fac        3          96
System.Runtime.CompilerServices.RuntimeHelpers+TryCode
73c95ddc      299        9568 System.IO.PathHelper
.shell: Process exited
```

Did you find it?

14. Try entering **.shell -ci "!dumpheap -stat" find "About"**

```
0:008> .shell -ci "!dumpheap -stat" find "About"
00b29d94        1         356 FotoVisionDesktop.AboutForm
.shell: Process exited
```

You found it—FotoVisionDesktop!FotoVision.AboutForm. One benefit of this angle is that, again, you don't have to be familiar with the source code to use this approach. Another benefit of this path is that it can produce a smaller output list than the "!dumpheap -stat" command.

One drawback of this approach is that it can still be troublesome to find your target class based on this "guessing" system but if you know the code it can work well.

You could also use the third party extension SosEx: SOSEX - Copyright 2007-2012 by Steve Johnson - http://www.stevestechspot.com/

It implements a command similar to X (in Windbg) called !sosex.mx

!sosex.mx *AboutForm*

<u>Finding the Help, About form, Method 3:</u>

```
0:004> .load sosex
0:004> !sosex.mx *AboutForm*
AppDomain 7503ea18 (Shared Domain)
---------------------------------------------------------
```

```
AppDomain 0098b638 (FotoVision.exe)
---------------------------------------------------------
module: FotoVision
  class: FotoVisionDesktop.AboutForm
    .ctor()
    Dispose(bool)
    InitializeComponent()
```

If you work with more than one extension (SosEx and Sos are Debugger Extensions) you can use
.chain to show all loaded extensions and their order and
.setdll to move one extension in front of all other extension

Finding the Help, About form, Method 4:

Finally, you can also use the **!Name2EE** command to obtain MT address and object instance address information. But unlike the **!dumpheap**, this command requires you to know the name of the class that you're looking for.

15. In this case, you can use the output of the previous step to get the parameter for the **!Name2EE** command.

```
0:004> !Name2EE FotoVision.exe FotoVisionDesktop.AboutForm
Module:      00802ed4
Assembly:    FotoVision.exe
Token:       02000002
MethodTable: 00b29d94
EEClass:     00b3fd58
Name:        FotoVisionDesktop.AboutForm
```

Do the Module and MethodTable address values match from the output of previous SOS commands?

16. Copy the EEClass address from the previous step and run the **!DumpClass** command on that address to view the members that compose this class.

Inspecting a MethodTable:

17. Run the SOS **!DumpMT** command and pass the MT address of the FotoVision.AboutForm class.

```
0:004> !dumpmt 00b29d94
EEClass:         00b3fd58
Module:          00802ed4
Name:            FotoVisionDesktop.AboutForm
mdToken:         02000002
File:            C:\LabFiles\FotoVision\4.0\C#\Desktop\Bin\FotoVision.exe)
BaseSize:        0x164
ComponentSize:   0x0
Slots in VTable: 377
Number of IFaces in IFaceMap: 21
```

How many instances of this class exist in the managed heap(s)? What is the SOS command that will give you this information?

Inspecting a MethodDesc:

18. Just as with the MethodTable in the previous step, dump all the MethodDescs of the FotoVisionDesktop.AboutForm class.

```
0:004> !dumpmt -md 00b29d94
EEClass:         00b3fd58
Module:          00802ed4
Name:            FotoVisionDesktop.AboutForm
mdToken:         02000002
File:            C:\LabFiles\FotoVision\4.0\C#\Desktop\Bin\FotoVision.exe)
BaseSize:        0x164
ComponentSize:   0x0
Slots in VTable: 377
Number of IFaces in IFaceMap: 21
MethodDesc Table
   Entry MethodDe    JIT Name
67f4f2d8 67de4e4c PreJIT System.Windows.Forms.Form.ToString()
73b85cf0 73888070 PreJIT System.Object.Equals(System.Object)
73b85920 73888090 PreJIT System.Object.GetHashCode()
72aa6730 7293e740 PreJIT System.ComponentModel.Component.Finalize()
73c2bdb0 7391f4c4 PreJIT System.MarshalByRefObject.GetLifetimeService()
73b63c20 7391f4cc PreJIT System.MarshalByRefObject.InitializeLifetimeService()
73bdbb7c 7391f4d4 PreJIT System.MarshalByRefObject.CreateObjRef(System.Type)
67f74000 67de576c PreJIT System.Windows.Forms.Control.get_CanRaiseEvents()
72aa5e78 7293e75c PreJIT
System.ComponentModel.Component.add_Disposed(System.EventHandler)
72ac6af8 7293e764 PreJIT
System.ComponentModel.Component.remove_Disposed(System.EventHandler)
67f77380 67de5cd4 PreJIT System.Windows.Forms.Control.get_Site()
67f1fb20 67de5cdc PreJIT
System.Windows.Forms.Control.set_Site(System.ComponentModel.ISite)
72aa65f0 7293e788 PreJIT System.ComponentModel.Component.Dispose()
0080e425 00b29d24   NONE FotoVisionDesktop.AboutForm.Dispose(Boolean)

extra output removed
…
```

19. Find the MT address of FotoVisionDesktop.AboutForm.InitializeComponent() and dump its MethodDesc (it's near the bottom of the list). If you don't remember the command to do this, use **!help** or review the SOS commands presented in this module.

```
!dumpmd 00b29d2c
Method Name:  FotoVisionDesktop.AboutForm.InitializeComponent()
Class:        00b3fd58
MethodTable:  00b29d94
mdToken:      06000003
Module:       00802ed4
IsJitted:     yes
CodeAddr:     04d2aab8
Transparency: Critical
```

Note the MethodDesc's metadata token. Remember that the first byte denotes the table type inside the assembly's metadata to which this token refers and the rest of the number is the row index for that item. In this case, the top byte is "06", which is the Table ID for a method table in the metadata.

20. *Can you determine at least how many rows (that is, the number of methods) are in the metadata method table for this assembly? In what row of the table does this method lie?*

Ending the debug session:

21. Type **q** in WinDBG's command line. This will end the debug session and shut down FotoVision.

Exercise 4: Determine if a Method Has Been JIT-Compiled or Not

In this exercise, you will learn how to determine if a method has been JIT-compiled (that is, has been called yet) or not. You learn where to set the breakpoint and what commands in SOS to use to assist in this operation.

Note: The exact output you will see will differ from the preceding, especially in memory addresses, method tables, method descriptions, code addresses etc.

Task Description

0.Start your debug session.

1. Open a new instance of **FotoVision.exe** from the <FotoVisionDir>\C#\Desktop\Bin directory.
2. On the Start menu, click Run. Then type **WinDBG -pn FotoVision.exe** to attach WinDBG to Fotovision.exe.

Finding a method that has not been JITted:

3. Load SOS.
4. You are looking for the MethodDesc of the method that runs whenever a user rotates a picture to the right. Pretend that you are the developer of FotoVision and that, while you don't remember the exact syntax of this method, you know it has "RotateRight_Click" in the name.

   Can you use the "!Name2EE" command to locate the class that contains the method? (notice that the parameter should be the full qualified name and case sensitive!)

```
0:005> 0:004> !Name2EE FotoVision!FotoVisionDesktop.MainForm
Module:      010e2ed4
Assembly:    FotoVision.exe
Token:       02000012
MethodTable: 010e71dc
EEClass:     054a5460
Name:        FotoVisionDesktop.MainForm
```

5. Next we can dump the list of methods for this form using **!DumpMT** and the MethodTable address from the previous command. To be even more targeted we can leverage the .shell command and find to filter our output:

```
0:005> .shell -ci "!dumpmt -md 010e71dc" find "RotateRight_Click"
010ecb20 010e7010   NONE
FotoVisionDesktop.MainForm.menuRotateRight_Click(System.Object, System.EventArgs)
.shell: Process exited
```

6. Dump the MethodDesc. Note: the second address (0x00237904) in the preceding output is the MethodDesc.

```
0:005> !dumpmd 00807010
Method Name:  FotoVisionDesktop.MainForm.menuRotateRight_Click(System.Object,
System.EventArgs)
Class:        00965460
MethodTable:  008071dc
mdToken:      06000102
Module:       00802ed4
IsJitted:     no
CodeAddr:     ffffffff
Transparency: Critical
```

Has this method been JIT-compiled yet? How can you tell?

Setting a breakpoint that will hit when your method has been JITed:

7. Use the "**!bpmd**" command to set a breakpoint on the menuRotateRight_Click method using the method name from the previous step.

```
0:005> !bpmd FotoVision.exe FotoVisionDesktop.MainForm.menuRotateRight_Click
Found 1 methods in module 00802ed4...
MethodDesc = 00807010
Adding pending breakpoints...
```

8. Type **g** to release control and run the application.

Calling the target method:

9. In FotoVision, in the **My Albums** section on the left-hand side, click the **Animals** album.
10. In the **Manage Photos** section, click the **picture of the bats.**
11. On the Edit menu, click **Rotate Photo** and then click **Rotate to the Right**. Be sure to use the menu and not the toolbar or else this will not work.

Note the breakpoint in WinDBG is hit.

Determining if the method has been JITted:

12. Type **!clrstack**. This is one of the ways to verify that you are in the expected location.
What method call is on the top of the stack?
13. Dump the MethodDesc of menuRotateRight_Click again.

```
0:000> !dumpmd 00807010
Method Name:  FotoVisionDesktop.MainForm.menuRotateRight_Click(System.Object,
System.EventArgs)
Class:        00965460
MethodTable:  008071dc
mdToken:      06000102
```

```
Module:       00802ed4
IsJitted:     yes
CodeAddr:     04d2f988
Transparency: Critical
```

Has this method been JITted, or do you need to step through a couple more lines of source before it will become JITted? How can you be sure of your answer?

Note the address of the CodeAddr from the previous step. Now let's look at the current native stack to find our JITted address on the top of the stack.

```
0:000> k
ChildEBP RetAddr
WARNING: Frame IP not in any known module. Following frames may be wrong.
0087e938 04faf777 0x4faf86f
*** WARNING: Unable to verify checksum for
C:\WINDOWS\assembly\NativeImages_v4.0.30319_32\System.Windows.Forms\70c6bf4a51d18b
4a9a1805cd48d1caad\System.Windows.Forms.ni.dll
0087ea20 68687438 0x4faf777
0087ea38 686882e1 System_Windows_Forms_ni+0x8b7438
0087ea44 685871eb System_Windows_Forms_ni+0x8b82e1
0087ea50 6858722a System_Windows_Forms_ni+0x7b71eb
0087ea58 68862b22 System_Windows_Forms_ni+0x7b722a
..
0087eb5c 67f75ed0 System_Windows_Forms_ni+0x1a5f89
0087ebe0 76008e71 System_Windows_Forms_ni+0x1a5ed0
0087ec0c 760090d1 USER32!_InternalCallWinProc+0x2b
0087eca0 7600a66f USER32!UserCallWinProcCheckWow+0x18e
0087ed0c 7600a6e0 USER32!DispatchMessageWorker+0x208
0087ed18 67fcb02c USER32!DispatchMessageW+0x10
0087ed54 67f84ac1 System_Windows_Forms_ni+0x1fb02c
..
0087eed8 74a02652 System_Windows_Forms_ni+0x196781
0087eee4 74a11580 clr!CallDescrWorkerInternal+0x34
0087ef38 74a1e670 clr!CallDescrWorkerWithHandler+0x6b
0087efb0 74b72955 clr!MethodDescCallSite::CallTargetWorker+0x152
0087f0d4 74b72891 clr!RunMain+0x1aa
0087f348 74b2ec8b clr!Assembly::ExecuteMainMethod+0x124
0087f848 74b2ed2e clr!SystemDomain::ExecuteMainMethod+0x63c
0087f8a0 74b2ee42 clr!ExecuteEXE+0x4c
0087f8e0 74b32370 clr!_CorExeMainInternal+0xdc
0087f91c 750aec94 clr!_CorExeMain+0x4d
0087f954 7512bbb7 mscoreei!_CorExeMain+0x10a
0087f95c 7512bbcc MSCOREE!_CorExeMain_Exported+0x77
0087f96c 75897c04 MSCOREE!_CorExeMain_Exported+0x8c
0087f980 77cbb90f KERNEL32!BaseThreadInitThunk+0x24
0087f9c8 77cbb8da ntdll!__RtlUserThreadStart+0x2f
0087f9d8 00000000 ntdll!_RtlUserThreadStart+0x1b …
```

14. Use the **!ip2md** command on the address at the top of the stack to verify what method is executing at the virtual address. Alternatively, you could use the **!ip2md** command on the value in the EIP register because you know EIP will always contain the currently executing instruction's address.

```
0:000> !ip2md 0x4faf86f
```

```
MethodDesc:   00a17010
Method Name:  FotoVisionDesktop.MainForm.menuRotateRight_Click(System.Object,
System.EventArgs)
Class:        023d5460
MethodTable:  00a171dc
mdToken:      06000102
Module:       00a12ed4
IsJitted:     yes
CodeAddr:     04faf850
Transparency: Critical Source file:
C:\LabFiles\FotoVision\4.0\C#\Desktop\Source\MainForm.cs @ 2512
```

Does the output from the "!ip2md" command display the expected method?

15. Unassemble the managed function from the address that you used in the **!ip2md** command to view the disassembly (**!U**). Note that the address is at the very beginning of the function. This makes sense, because that's where you set the breakpoint.

```
0:000> !u 0x4faf86f
Normal JIT generated code
FotoVisionDesktop.MainForm.menuRotateRight_Click(System.Object, System.EventArgs)
Begin 04faf850, size 51

C:\LabFiles\FotoVision\4.0\C#\Desktop\Source\MainForm.cs @ 2512:
04faf850 55              push    ebp
04faf851 8bec            mov     ebp,esp
04faf853 83ec0c          sub     esp,0Ch
04faf856 33c0            xor     eax,eax
04faf858 8945f4          mov     dword ptr [ebp-0Ch],eax
04faf85b 894df8          mov     dword ptr [ebp-8],ecx
04faf85e 8955fc          mov     dword ptr [ebp-4],edx
04faf861 833d7831a10000  cmp     dword ptr ds:[0A13178h],0
04faf868 7405            je      04faf86f
04faf86a e8a1cad86f      call    clr!JIT_DbgIsJustMyCode (74d3c310)
>>> 04faf86f 90              nop

C:\LabFiles\FotoVision\4.0\C#\Desktop\Source\MainForm.cs @ 2513:
04faf870 b924a4da04      mov     ecx,4DAA424h (MT:
FotoVisionDesktop.ActionEventArgs)
04faf875 e85a28a5fb      call    00a020d4 (JitHelp: CORINFO_HELP_NEWSFAST)
04faf87a 8945f4          mov     dword ptr [ebp-0Ch],eax
04faf87d 8b4df4          mov     ecx,dword ptr [ebp-0Ch]
04faf880 ba01000000      mov     edx,1
04faf885 ff1514a4da04    call    dword ptr ds:[4DAA414h]
(FotoVisionDesktop.ActionEventArgs..ctor(FotoVisionDesktop.PhotoAction), mdToken:
060002b8)
04faf88b ff75f4          push    dword ptr [ebp-0Ch]
04faf88e 8b4df8          mov     ecx,dword ptr [ebp-8]
04faf891 8b55f8          mov     edx,dword ptr [ebp-8]
04faf894 e847d1a6fb      call    00a1c9e0
(FotoVisionDesktop.MainForm.paneDetails_Action(System.Object,
FotoVisionDesktop.ActionEventArgs), mdToken: 0600011a)
04faf899 90              nop

C:\LabFiles\FotoVision\4.0\C#\Desktop\Source\MainForm.cs @ 2514:
04faf89a 90              nop
04faf89b 8be5            mov     esp,ebp
04faf89d 5d              pop     ebp
```

```
04faf89e c20400          ret     4
```

16. Release control so that the application can run.
17. In the WinDBG command line, type **qd** to detach the debugger and allow FotoVision to remain alive.
    You may find this command handy in cases where you want to debug an application while allowing for the debugger to detach without having to restart the application.