.NET Debugging – Exercises

# Exercise 1

In this exercise, you will use a crash dump to diagnose a crashing application.

1. Run the FileExplorer.exe application from the Ch5\_FileExplorerCrash\Binaries folder.
2. Click on a directory in the directory tree on the left. The application crashes.
3. Use ADPlus (from the Debugging Tools for Windows package) to capture a crash dump of the application. Use the –crash command line switch and the –pn parameter to configure the process name to monitor.
4. Use WinDbg to open the dump file generated by ADPlus. By default it’s stored in a subdirectory of the ADPlus installation directory.
5. Key in the following commands:

*~0s*

*.loadby sos mscorwks*

*!CLRStack -a*

*!PrintException*

1. What is the exception that was thrown? What is the call stack?
2. Optional: Use .NET Reflector to open the FileExplorer.exe application and look for the underlying cause for the exception you’ve experienced.

# Exercise 2

In this exercise, you will use a hang dump to diagnose a hanging application.

1. Run the FileExplorer.exe application from the Ch5\_FileExplorerHang\Binaries folder.
2. Click on a directory in the directory tree on the left. If the list view on the right is empty, choose another directory (specifically, one with files in it).
3. Double click one of the files in the list view on the right. The application hangs—the UI stops responding and there is no visible progress. (What is supposed to happen is that a Notepad window should open with the text of the file you double-clicked.)
4. Use ADPlus to capture a hang dump of the application. Use the –hang command line switch and the –pn parameter to configure the process name to capture.
5. Use WinDbg to open the dump file generated by ADPlus. By default it’s stored in a subdirectory of the ADPlus installation directory.
6. Key in the following commands:

*~0s*

*.loadby sos mscorwks*

*!CLRStack -a*

1. What is the application’s main thread busy doing?
2. Optional: Use .NET Reflector to open the FileExplorer.exe application and look for the underlying cause for the hang you’ve experienced.

# Exercise 3

In this exercise, you will use SOS.DLL to diagnose a managed memory leak.

1. Run the FileExplorer.exe application from the Ch5\_FileExplorerHang\Binaries folder.
2. Run Performance Monitor and watch the counter called .NET CLR Memory\# Bytes in All Heaps for the FileExplorer.exe process.
3. Click on a directory in the directory tree on the left. If the list view on the right is empty, choose another directory (specifically, one with files in it).
4. Repeat step 2 multiple times for different directories and watch the memory usage performance counter in the meantime. The application is leaking managed memory.
5. Use WinDbg to attach to the application’s process (File🡪Attach to a Process).
6. Key in the following commands:

*.loadby sos mscorwks*

*!dumpheap –stat*

1. Proceed to analyze the application’s memory usage. Use commands such as !dumpheap and !gcroot to detect which objects are being unnecessarily retained and how.
2. Optional: Use .NET Reflector to open the FileExplorer.exe application and look for the underlying cause for the memory leak you’ve experienced.