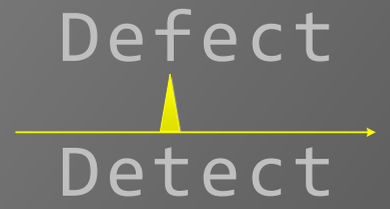


Windows Debugging[4]

Accelerated

Version 4.0

Dmitry Vostokov
Software Diagnostics Services

Published by OpenTask, Republic of Ireland

OpenTask books and magazines are available through booksellers and distributors worldwide. For further information or comments, send requests to press@opentask.com.

A CIP catalog record for this book is available from the British Library.

ISBN-l3: 978-1912636-72-3 (Paperback)

Revision 4.00 (February 2024)

Contents

Exercise UD1

Goal: Learn how code generation parameters can influence process execution behavior.

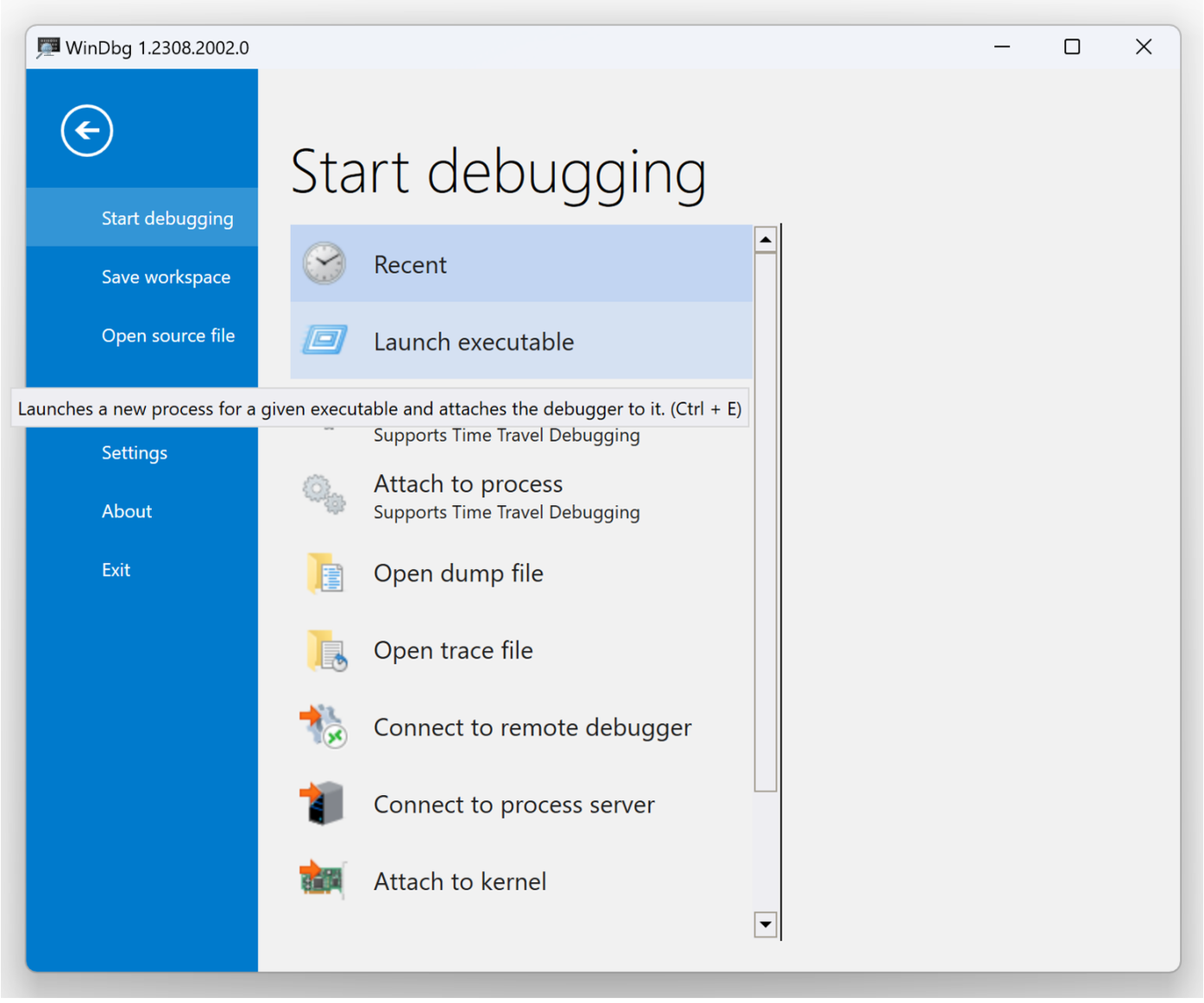
Elementary Diagnostics Patterns: Error Message or Crash.

Memory Analysis Patterns: Message Box or Exception Stack Trace; Constant Subtrace.

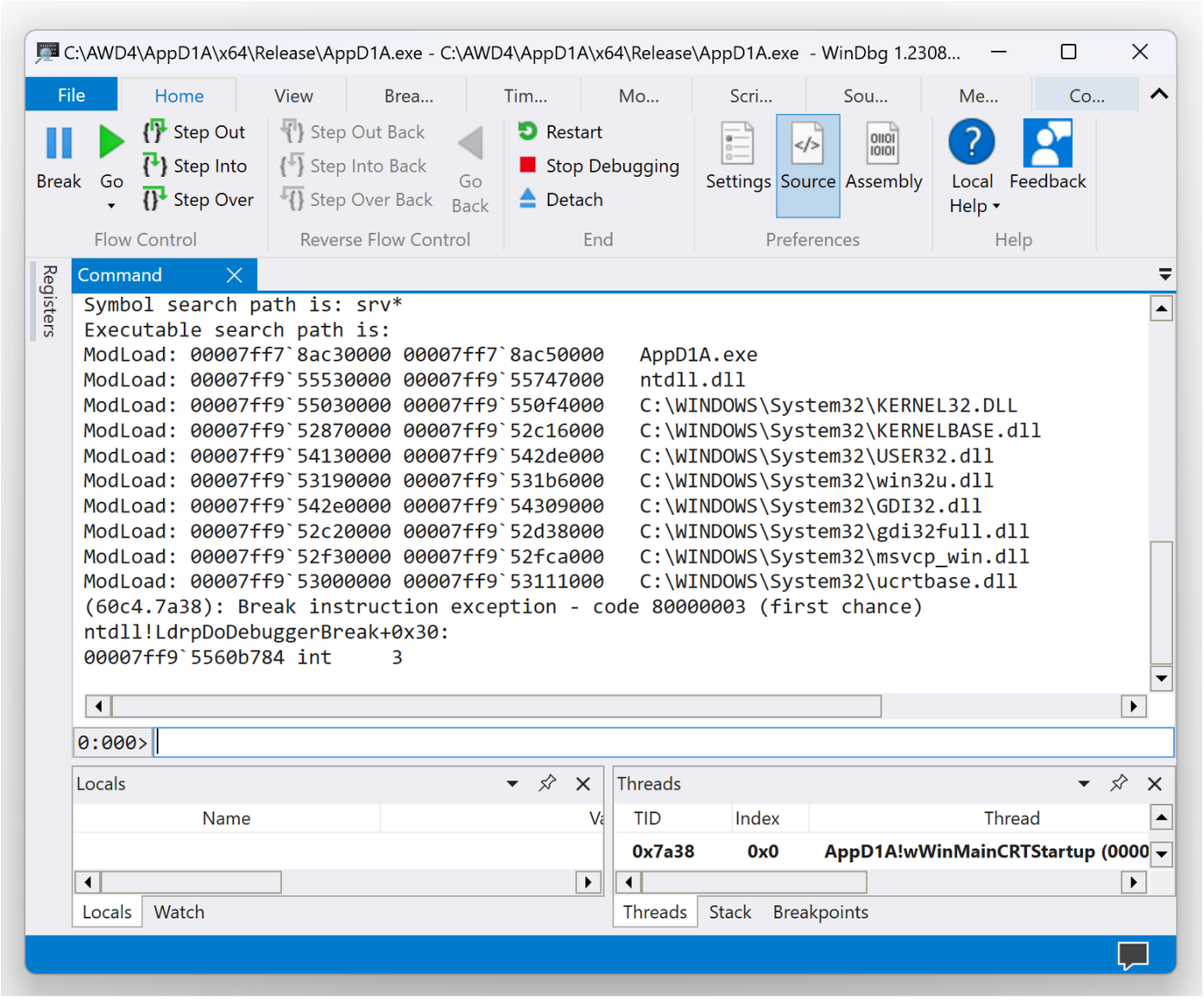
Debugging Implementation Patterns: Break-in; Scope; Variable Value; Type Structure; Code Breakpoint.

1. Launch WinDbg.

2. Open *\AWD4\AppD1A\x64\Release\AppD1A.exe* executable by choosing *File* \ *Launch executable* menu option:



3. We get the executable file loaded and ready for a debugging session:



From now on, we only show the output from the command window unless we need another view.

```
Microsoft (R) Windows Debugger Version 10.0.25921.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: C:\AWD4\AppD1A\x64\Release\AppD1A.exe

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
ModLoad: 00007ff7`8ac30000 00007ff7`8ac50000   AppD1A.exe
ModLoad: 00007ff9`55530000 00007ff9`55747000   ntdll.dll
ModLoad: 00007ff9`55030000 00007ff9`550f4000   C:\WINDOWS\System32\KERNEL32.DLL
ModLoad: 00007ff9`52870000 00007ff9`52c16000   C:\WINDOWS\System32\KERNELBASE.dll
ModLoad: 00007ff9`54130000 00007ff9`542de000   C:\WINDOWS\System32\USER32.dll
ModLoad: 00007ff9`53190000 00007ff9`531b6000   C:\WINDOWS\System32\win32u.dll
ModLoad: 00007ff9`542e0000 00007ff9`54309000   C:\WINDOWS\System32\GDI32.dll
ModLoad: 00007ff9`52c20000 00007ff9`52d38000   C:\WINDOWS\System32\gdi32full.dll
ModLoad: 00007ff9`52f30000 00007ff9`52fca000   C:\WINDOWS\System32\msvcp_win.dll
ModLoad: 00007ff9`53000000 00007ff9`53111000   C:\WINDOWS\System32\ucrtbase.dll
(60c4.7a38): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007ff9`5560b784 int     3
```

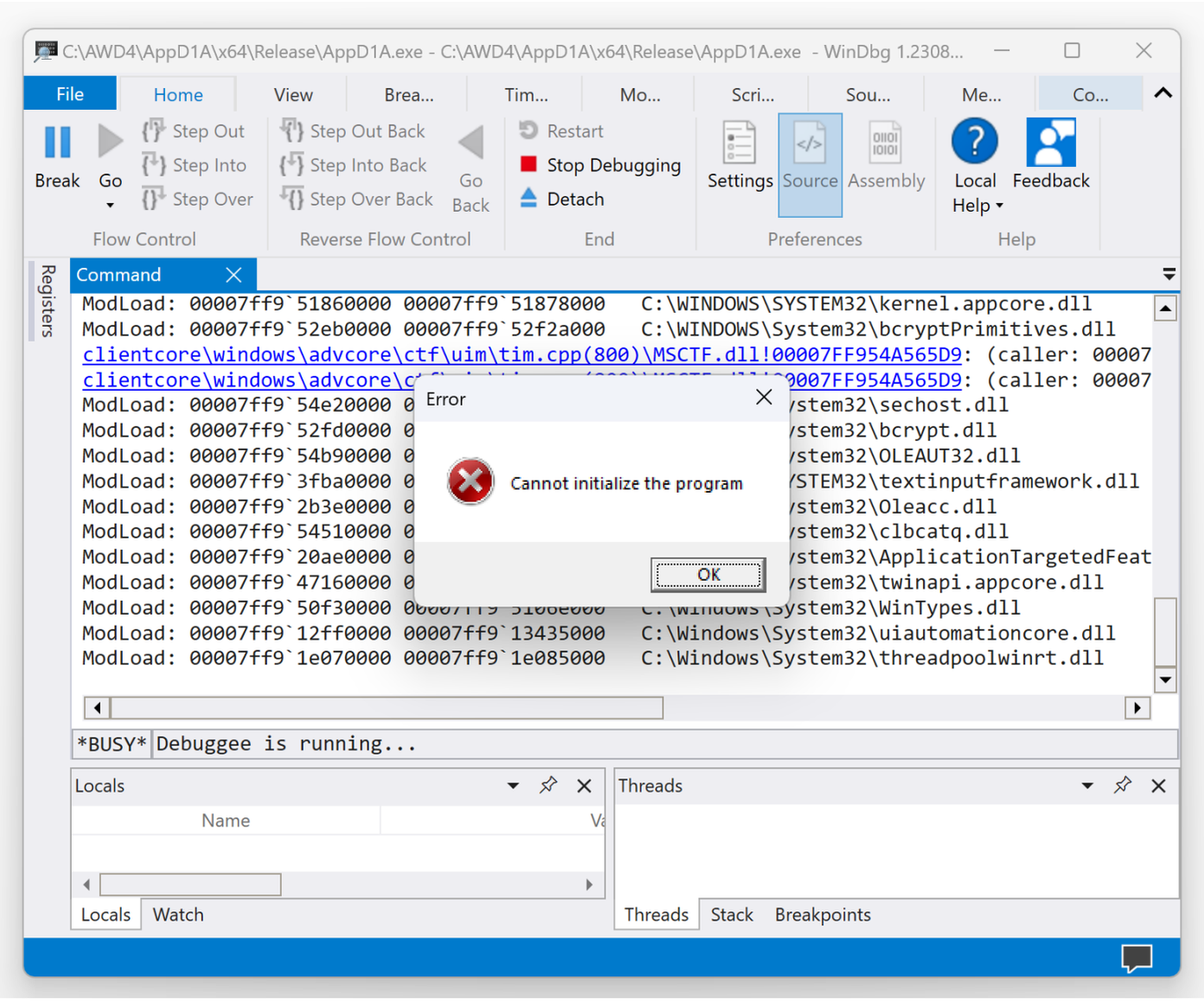
4. Open a log file (useful when the output doesn't fit into the buffer and we need to search for something):

```
0:000> .logopen C:\AWD4\D1A.log
Opened log file 'C:\AWD4\D1A.log'
```

5. The **lm** command lists loaded modules and their addresses (it also shows whether symbols files are loaded):

```
0:000> lm
start             end                 module name
00007ff7`8ac30000 00007ff7`8ac50000   AppD1A   C (private pdb symbols)
C:\ProgramData\Dbg\sym\AppD1A.pdb\926ED291EC994719BD0DFB167D2EB42E1\AppD1A.pdb
00007ff9`52870000 00007ff9`52c16000   KERNELBASE   (deferred)
00007ff9`52c20000 00007ff9`52d38000   gdi32full   (deferred)
00007ff9`52f30000 00007ff9`52fca000   msvcp_win   (deferred)
00007ff9`53000000 00007ff9`53111000   ucrtbase   (deferred)
00007ff9`53190000 00007ff9`531b6000   win32u     (deferred)
00007ff9`54130000 00007ff9`542de000   USER32     (deferred)
00007ff9`542e0000 00007ff9`54309000   GDI32      (deferred)
00007ff9`55030000 00007ff9`550f4000   KERNEL32   (deferred)
00007ff9`55530000 00007ff9`55747000   ntdll      (pdb symbols)
C:\ProgramData\Dbg\sym\ntdll.pdb\58A282C24AEE7E03A8CF8CB0A782CE0C1\ntdll.pdb
```

6. We continue process execution using the **g** command. We may get an error message box:



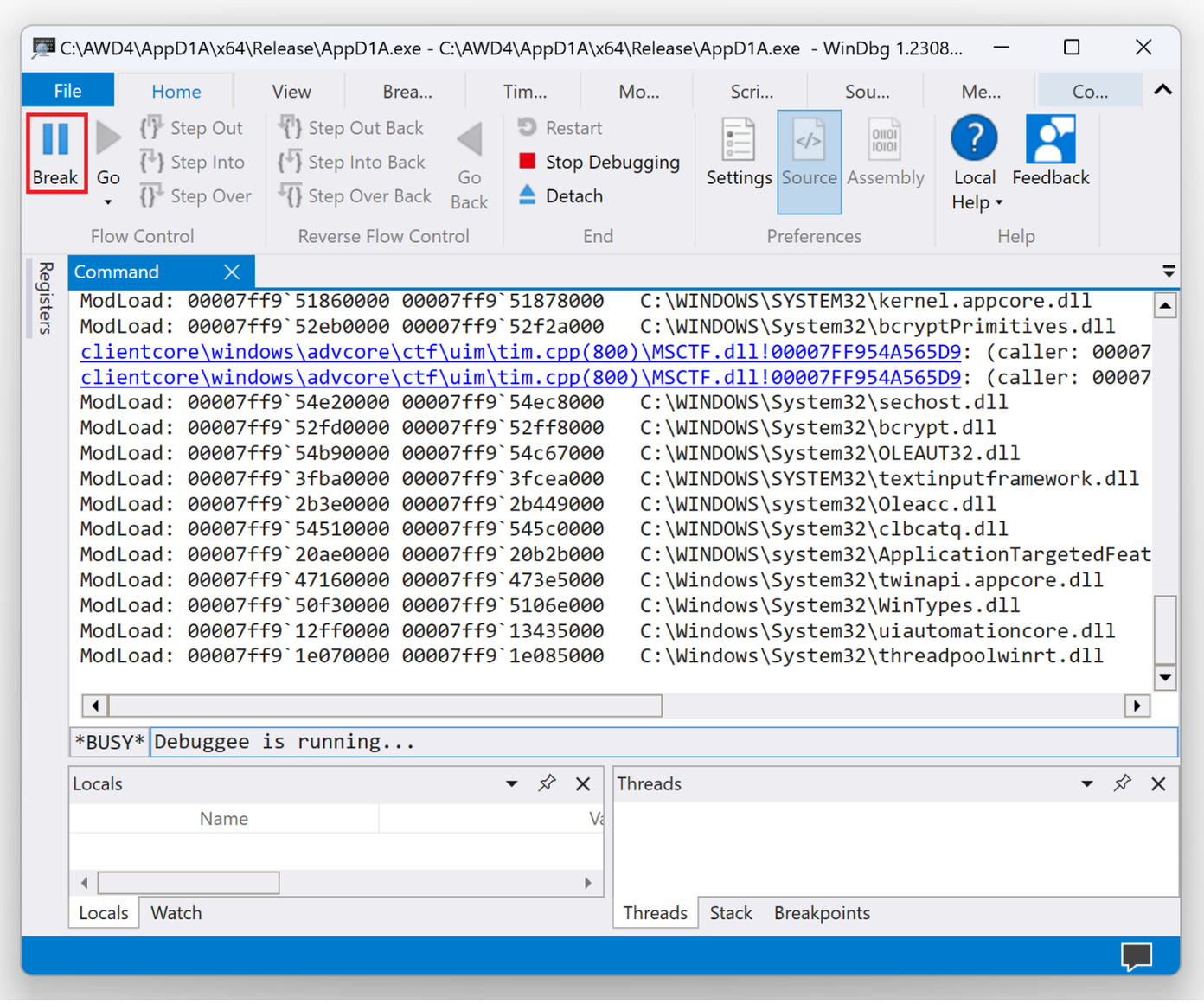
Note: Instead of an error message box, we may get exceptions. In such a case, we have to ignore any first chance exceptions until we come to a second chance exception (the crash scenario), for example:

```
0:000> g
ModLoad: 00007ff9`6caf0000 00007ff9`6cb21000   C:\WINDOWS\System32\IMM32.DLL
(1588.7e98): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
USER32!StringDuplicateW+0x20:
00007ff9`6dfd4864 66392c41        cmp     word ptr [rcx+rax*2],bp ds:0bc8acf0`00000000=????

0:000> g
(1588.7e98): Access violation - code c0000005 (!!! second chance !!!)
USER32!StringDuplicateW+0x20:
00007ff9`6dfd4864 66392c41        cmp     word ptr [rcx+rax*2],bp ds:0bc8acf0`00000000=????
```

There, we see that the crash happens in the **USER32** module with the following CPU state:

```
0:000> r
rax=0000000000000000 rbx=000000cc248ff4c0 rcx=0bc8acf000000000
rdx=0bc8acf000000000 rsi=000000cc248ff450 rdi=0bc8acf000000000
rip=00007ff96dfd4864 rsp=000000cc248ff350 rbp=0000000000000000
 r8=000000cc248ff4c0  r9=0000000000000080 r10=000001d56e280000
r11=0000000000000000 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up ei pl zr ac po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010254
USER32!StringDuplicateW+0x20:
00007ff9`6dfd4864 66392c41        cmp     word ptr [rcx+rax*2],bp ds:0bc8acf0`00000000=????
```

7. In the case of an error message box, we break in using the *Break* button:




```
(60c4.356c): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ff9`555d3060 int     3
```

8. We switch to the original thread #0 and check its stack trace:

```
0:001> ~0s
win32u!NtUserWaitMessage+0x14:
00007ff9`531915f4 ret

0:000> kL
 # Child-SP          RetAddr               Call Site
00 000000e9`d54ff5a8 00007ff9`541316c6     win32u!NtUserWaitMessage+0x14
01 000000e9`d54ff5b0 00007ff9`5413152b     USER32!DialogBox2+0x172
02 000000e9`d54ff660 00007ff9`541aa666     USER32!InternalDialogBox+0x127
03 000000e9`d54ff6c0 00007ff9`541a8fc9     USER32!SoftModalMessageBox+0x826
04 000000e9`d54ff800 00007ff9`541a9da8     USER32!MessageBoxWorker+0x341
05 000000e9`d54ff9b0 00007ff9`541a9e2e     USER32!MessageBoxTimeoutW+0x198
06 000000e9`d54ffab0 00007ff7`8ac3107f     USER32!MessageBoxW+0x4e
07 000000e9`d54ffaf0 00007ff7`8ac3168a     AppD1A!wWinMain+0x7f
08 (Inline Function) --------`--------     AppD1A!invoke_main+0x21
09 000000e9`d54ffb50 00007ff9`5504257d     AppD1A!__scrt_common_main_seh+0x106
0a 000000e9`d54ffb90 00007ff9`5558aa58     KERNEL32!BaseThreadInitThunk+0x1d
0b 000000e9`d54ffbc0 00000000`00000000     ntdll!RtlUserThreadStart+0x28
```

9. We also set the source search path:

```
0:000> .srcpath+ C:\AWD4\AppD1A\AppD1A
Source search path is: SRV*;C:\AWD4\AppD1A\AppD1A

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       SRV*
OK                                             C:\AWD4\AppD1A\AppD1A
```

Note: If you get a second chance exception (the crash scenario) instead of an error message box, then the default analysis command gives us the source code:

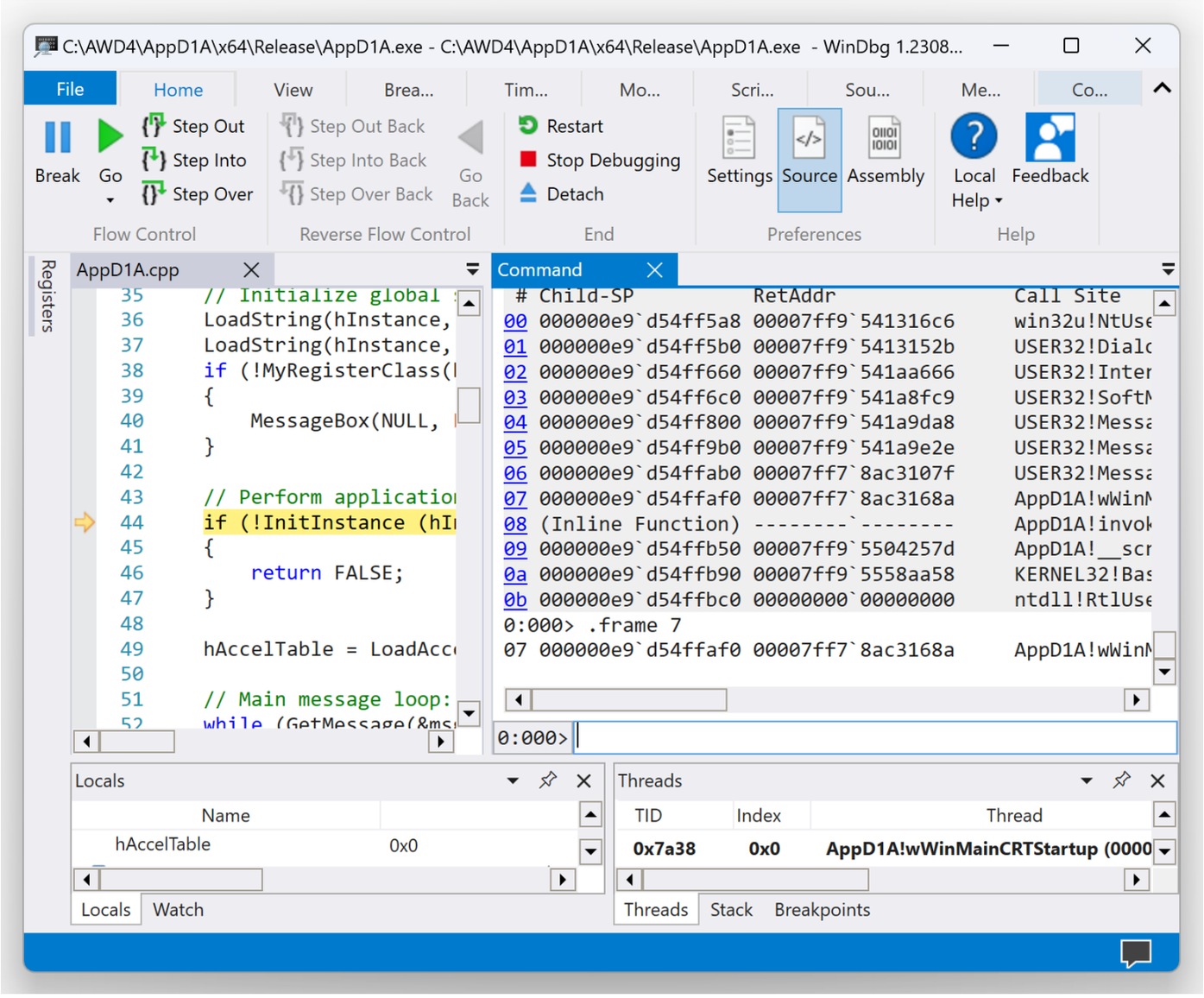
```
0:000> !analyze -v
*******************************************************************************
*                                                                             *
*                        Exception Analysis                                   *
*                                                                             *
*******************************************************************************


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Read

    Key  : Analysis.CPU.mSec
    Value: 1093

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 1811

    Key  : Analysis.Init.CPU.mSec
    Value: 2749

    Key  : Analysis.Init.Elapsed.mSec
    Value: 1229233

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 89

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 698874

    Key  : Timeline.Process.Start.DeltaSec
    Value: 1228

    Key  : WER.OS.Branch
    Value: co_release

    Key  : WER.OS.Timestamp
```

```
    Value: 2021-06-04T16:28:00Z

    Key  : WER.OS.Version
    Value: 10.0.22000.1


NTGLOBALFLAG:  470

PROCESS_BAM_CURRENT_THROTTLED: 0

PROCESS_BAM_PREVIOUS_THROTTLED: 0

APPLICATION_VERIFIER_FLAGS:  0

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 00007ff96dfd4864 (USER32!StringDuplicateW+0x0000000000000020)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 0000000000000000
   Parameter[1]: ffffffffffffffff
Attempt to read from address ffffffffffffffff

FAULTING_THREAD:  00007e98

PROCESS_NAME:  AppD1A.exe

READ_ADDRESS:  ffffffffffffffff

ERROR_CODE: (NTSTATUS) 0xc0000005 - The instruction at 0x%p referenced memory at 0x%p. The memory could not be %s.

EXCEPTION_CODE_STR:  c0000005

EXCEPTION_PARAMETER1:  0000000000000000

EXCEPTION_PARAMETER2:  ffffffffffffffff

STACK_TEXT:
000000cc`248ff350 00007ff9`6dfd7fda     : 000000cc`248ff4c0 0bc8acf0`00000000 000000cc`248ff450 00000000`00000000 : USER32!StringDuplicateW+0x20
000000cc`248ff380 00007ff9`6dfd7d6b     : 000000cc`248ff760 000000cc`248ff4d0 00000000`00000000 00007ff9`6dff5a00 : USER32!InitClsMenuNameW+0x76
000000cc`248ff3d0 00007ff9`6dfd7c49     : 000000cc`248ff810 00000000`00000000 00000000`00000000 00000000`00000000 : USER32!RegisterClassExWOWW+0x113
000000cc`248ff730 00007ff6`0bc7116d     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : USER32!RegisterClassW+0x59
000000cc`248ff7c0 00007ff6`0bc7105c     : 00007ff6`0bc70000 00000000`00000000 00000000`00000000 00000000`00000000 : AppD1A!MyRegisterClass+0x8d
000000cc`248ff840 00007ff6`0bc7166a     : 00007ff6`0bc70000 00000000`00000000 000001d5`6deb529e 00000000`0000000a : AppD1A!wWinMain+0x5c
000000cc`248ff8a0 00007ff9`6d8c54e0     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : AppD1A!__scrt_common_main_seh+0x106
000000cc`248ff8e0 00007ff9`6e44485b     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : KERNEL32!BaseThreadInitThunk+0x10
000000cc`248ff910 00000000`00000000     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x2b


STACK_COMMAND:  ~0s ; .cxr ; kb

FAULTING_SOURCE_LINE:  C:\AWD3\AppD1A\AppD1A\AppD1A.cpp

FAULTING_SOURCE_FILE:  C:\AWD3\AppD1A\AppD1A\AppD1A.cpp

FAULTING_SOURCE_LINE_NUMBER:  84

FAULTING_SOURCE_CODE:
    80:    wc.lpszMenuName                 = MAKEINTRESOURCE(IDC_APPD1A);
    81:    wc.lpszClassName      = szWindowClass;
    82:
    83:    return RegisterClass(&wc);
>   84: }
    85:
    86: //
    87: //   FUNCTION: InitInstance(HINSTANCE, int)
    88: //
    89: //   PURPOSE: Saves instance handle and creates main window


SYMBOL_NAME:  appd1a!MyRegisterClass+8d

MODULE_NAME: AppD1A

IMAGE_NAME:  AppD1A.exe

FAILURE_BUCKET_ID:  INVALID_POINTER_READ_c0000005_AppD1A.exe!MyRegisterClass

OS_VERSION:  10.0.22000.1

BUILDLAB_STR:  co_release

OSPLATFORM_TYPE:  x64

OSNAME:  Windows 10

FAILURE_ID_HASH:  {0e59b433-475d-53b5-9229-de642189649b}

Followup:     MachineOwner
---------
```

10. In the case of an error message box, we switch to the *WinMain* thread stack frame:

```
0:000> .frame 7
07 000000e9`d54ffaf0 00007ff7`8ac3168a     AppD1A!wWinMain+0x7f [C:\AWD4\AppD1A\AppD1A\AppD1A.cpp @ 44]
```

Note: We see a source code window immediately to the left of the command window:



Note: In the case of a second-chance exception (the crash scenario), we get a similar stack trace and also set the appropriate stack frame in our code that called Windows API:

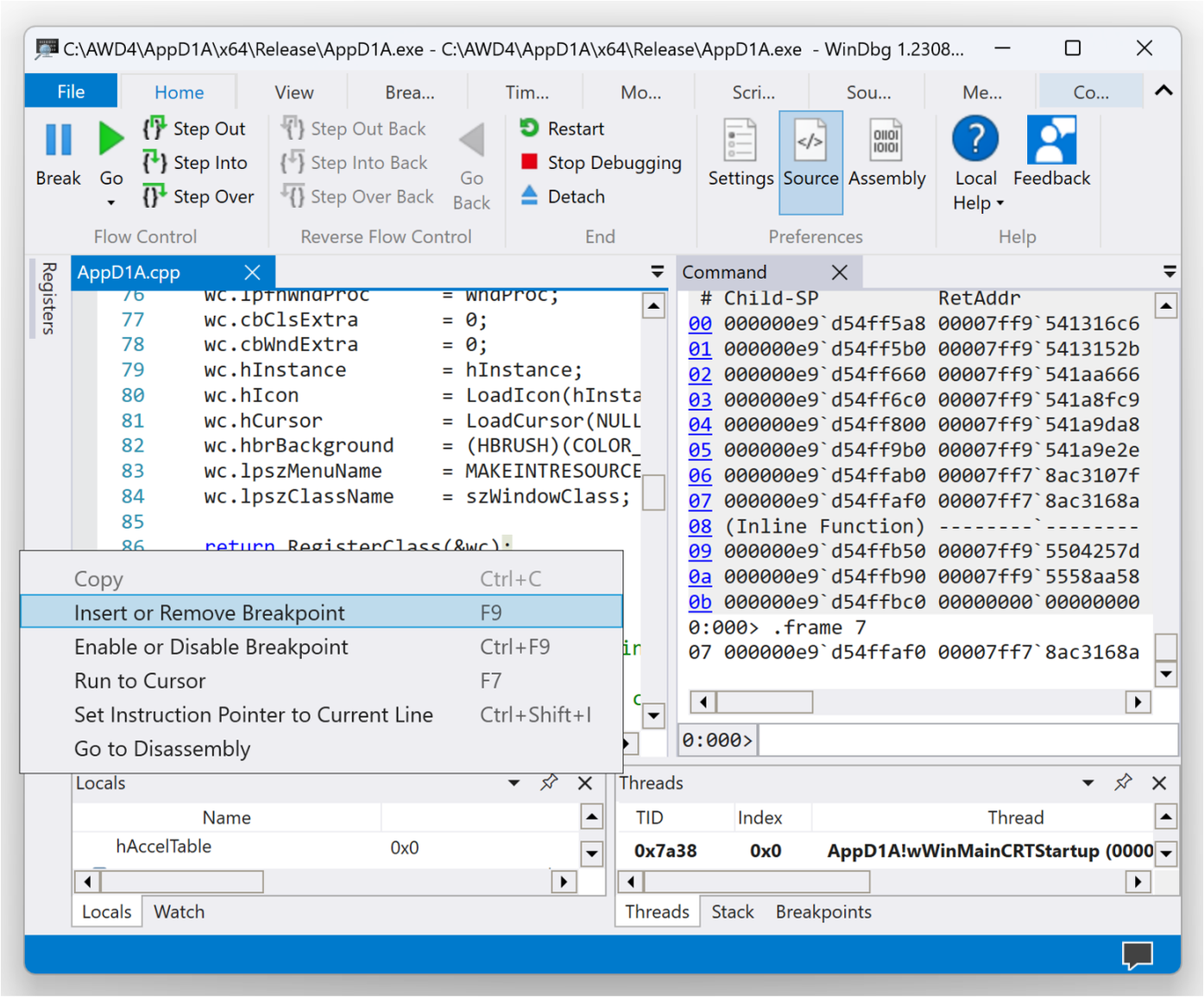
```
0:000> k
 # Child-SP          RetAddr               Call Site
00 000000cc`248ff350 00007ff9`6dfd7fda     USER32!StringDuplicateW+0x20
01 000000cc`248ff380 00007ff9`6dfd7d6b     USER32!InitClsMenuNameW+0x76
02 000000cc`248ff3d0 00007ff9`6dfd7c49     USER32!RegisterClassExWOWW+0x113
03 000000cc`248ff730 00007ff6`0bc7116d     USER32!RegisterClassW+0x59
04 000000cc`248ff7c0 00007ff6`0bc7105c     AppD1A!MyRegisterClass+0x8d [C:\AWD3\AppD1A\AppD1A\AppD1A.cpp @ 84]
05 000000cc`248ff840 00007ff6`0bc7166a     AppD1A!wWinMain+0x5c [C:\AWD3\AppD1A\AppD1A\AppD1A.cpp @ 41]
06 (Inline Function) --------`--------     AppD1A!invoke_main+0x21
[d:\a01\_work\43\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 118]
07 000000cc`248ff8a0 00007ff9`6d8c54e0     AppD1A!__scrt_common_main_seh+0x106
[d:\a01\_work\43\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 288]
08 000000cc`248ff8e0 00007ff9`6e44485b     KERNEL32!BaseThreadInitThunk+0x10
```

```
09 000000cc`248ff910 00000000`00000000     ntdll!RtlUserThreadStart+0x2b

0:000> .frame 4
04 000000cc`248ff7c0 00007ff6`0bc7105c     AppD1A!MyRegisterClass+0x8d [C:\AWD3\AppD1A\AppD1A\AppD1A.cpp @ 84]
```

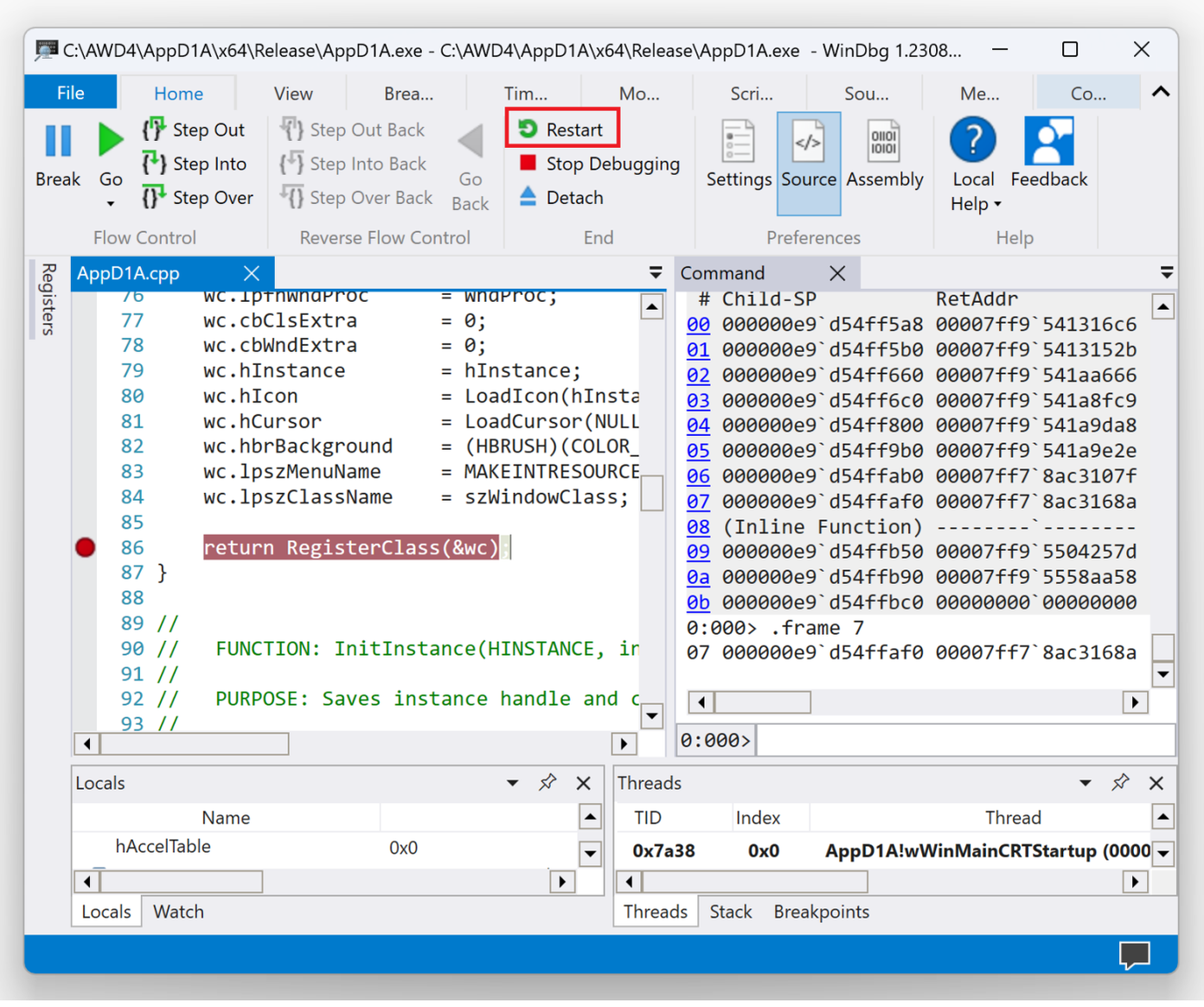
11. The yellow source code highlighting points to the next line to be executed if we dismiss the message box. Therefore, we need to investigate the previous call to the *MyRegisterClass* function. Inside, it returns the result of the *RegisterClass* Windows API call:

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASS wc;

	wc.style		= CS_HREDRAW | CS_VREDRAW;
	wc.lpfnWndProc	= WndProc;
	wc.cbClsExtra	= 0;
	wc.cbWndExtra	= 0;
	wc.hInstance	= hInstance;
	wc.hIcon		= LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPD1A));
	wc.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wc.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wc.lpszMenuName	= MAKEINTRESOURCE(IDC_APPD1A);
	wc.lpszClassName	= szWindowClass;

	return RegisterClass(&wc);
}
```

We put a code breakpoint at the line number #86 (right-click or F9):



Then we restart our debugging session:

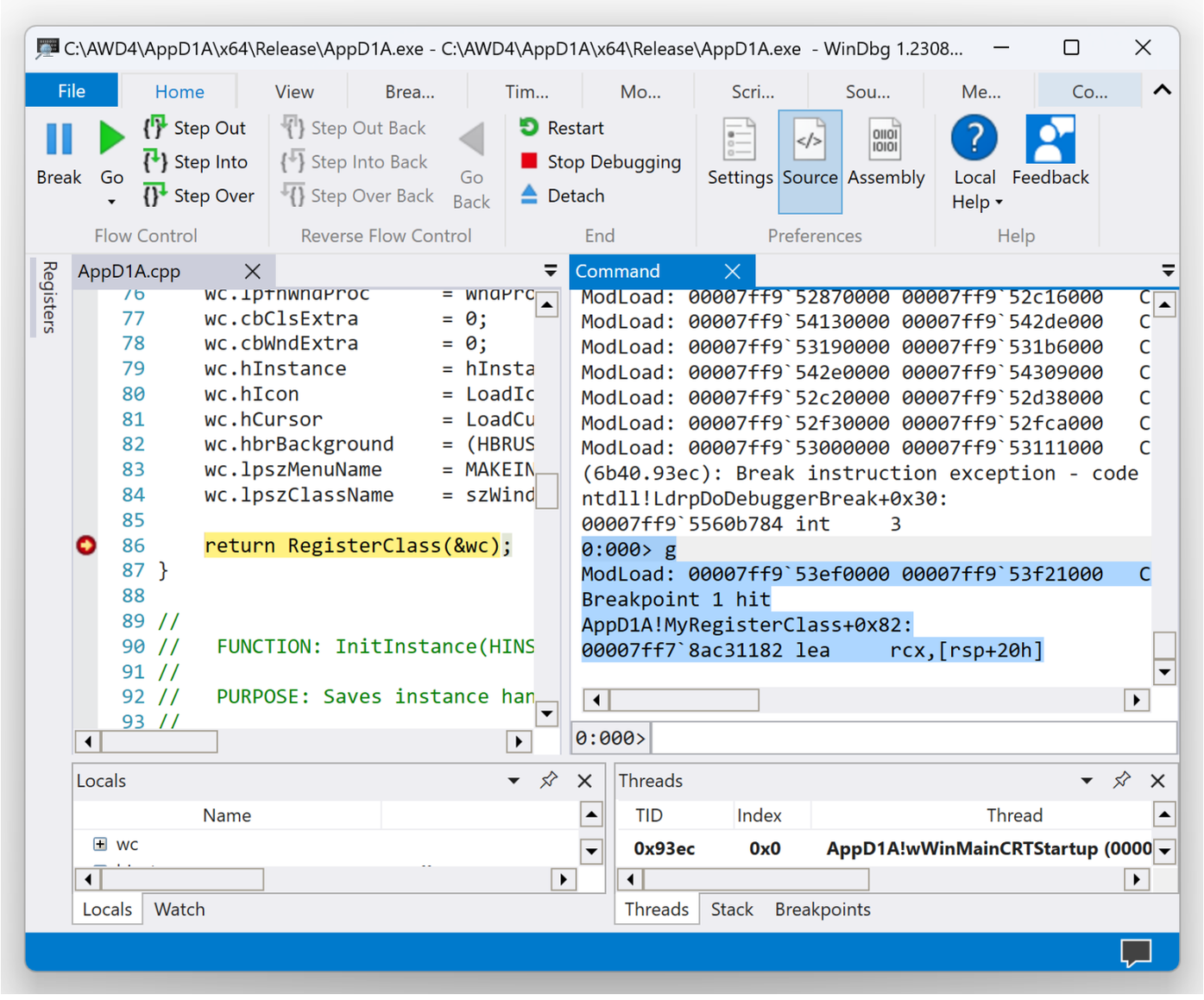



```
Microsoft (R) Windows Debugger Version 10.0.25921.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: C:\AWD4\AppD1A\x64\Release\AppD1A.exe

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
ModLoad: 00007ff7`8ac30000 00007ff7`8ac50000   AppD1A.exe
ModLoad: 00007ff9`55530000 00007ff9`55747000   ntdll.dll
ModLoad: 00007ff9`55030000 00007ff9`550f4000   C:\WINDOWS\System32\KERNEL32.DLL
ModLoad: 00007ff9`52870000 00007ff9`52c16000   C:\WINDOWS\System32\KERNELBASE.dll
ModLoad: 00007ff9`54130000 00007ff9`542de000   C:\WINDOWS\System32\USER32.dll
ModLoad: 00007ff9`53190000 00007ff9`531b6000   C:\WINDOWS\System32\win32u.dll
ModLoad: 00007ff9`542e0000 00007ff9`54309000   C:\WINDOWS\System32\GDI32.dll
ModLoad: 00007ff9`52c20000 00007ff9`52d38000   C:\WINDOWS\System32\gdi32full.dll
ModLoad: 00007ff9`52f30000 00007ff9`52fca000   C:\WINDOWS\System32\msvcp_win.dll
```

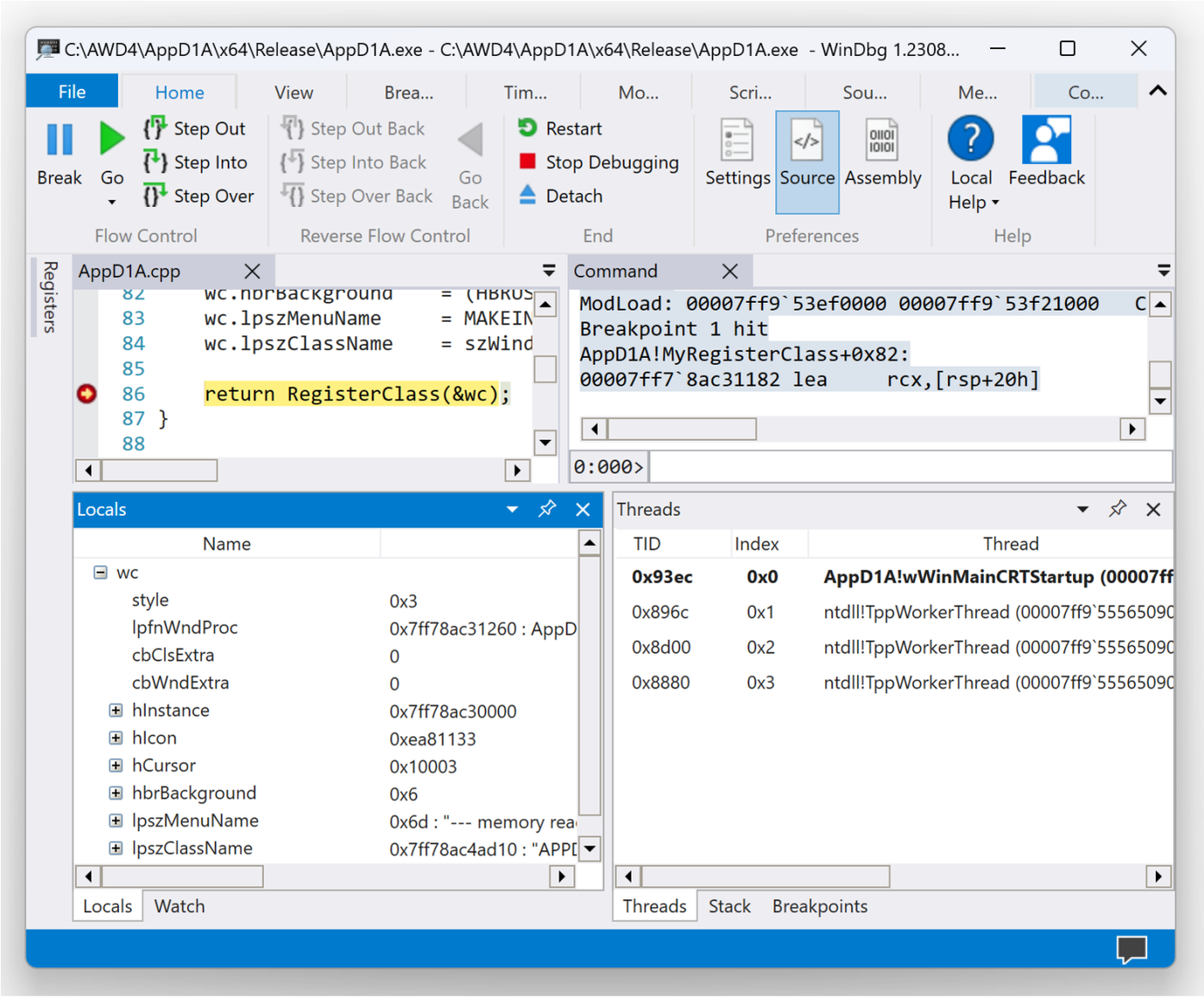
```
ModLoad: 00007ff9`53000000 00007ff9`53111000   C:\WINDOWS\System32\ucrtbase.dll
(6b40.93ec): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007ff9`5560b784 int     3
```

12. We then resume execution until we hit the breakpoint:

```
0:000> g
ModLoad: 00007ff9`53ef0000 00007ff9`53f21000   C:\WINDOWS\System32\IMM32.DLL
Breakpoint 1 hit
AppD1A!MyRegisterClass+0x82:
00007ff7`8ac31182 lea     rcx,[rsp+20h]
```



13. We can now expand local structures in the *Locals* window (for example, *wc*):



We can also dump this variable using type information:

```
0:000> dt wc
Local var @ 0xe17d2ff760 Type tagWNDCLASSW
   +0x000 style            : 3
   +0x004 lpfnWndProc      : 0x00007ff7`8ac31260     int64  AppD1A!WndProc+0
   +0x00c cbClsExtra       : 0n0
   +0x010 cbWndExtra       : 0n0
   +0x014 hInstance        : 0x00007ff7`8ac30000 HINSTANCE__
   +0x01c hIcon            : 0x00000000`0ea81133 HICON__
   +0x024 hCursor          : 0x00000000`00010003 HICON__
   +0x02c hbrBackground    : 0x00000000`00000006 HBRUSH__
   +0x034 lpszMenuName     : 0x00000000`0000006d  "--- memory read error at address 0x00000000`0000006d -
--"
   +0x03c lpszClassName    : 0x00007ff7`8ac4ad10  "APPD1A"
```

14. We can also list all other local variables and parameters for the current frame:

```
0:000> dv /i /V
prv param  000000e1`7d2ff7c0 @rsp+0x0080             hInstance = 0x00007ff7`8ac30000
prv local  000000e1`7d2ff760 @rsp+0x0020                    wc = struct tagWNDCLASSW
```

Note: Since all structure members seem to be valid (*lpszMenuName* read error can be ignored because in the source code, it is specified as an integer resource index), let's compare it with another application that doesn't crash.

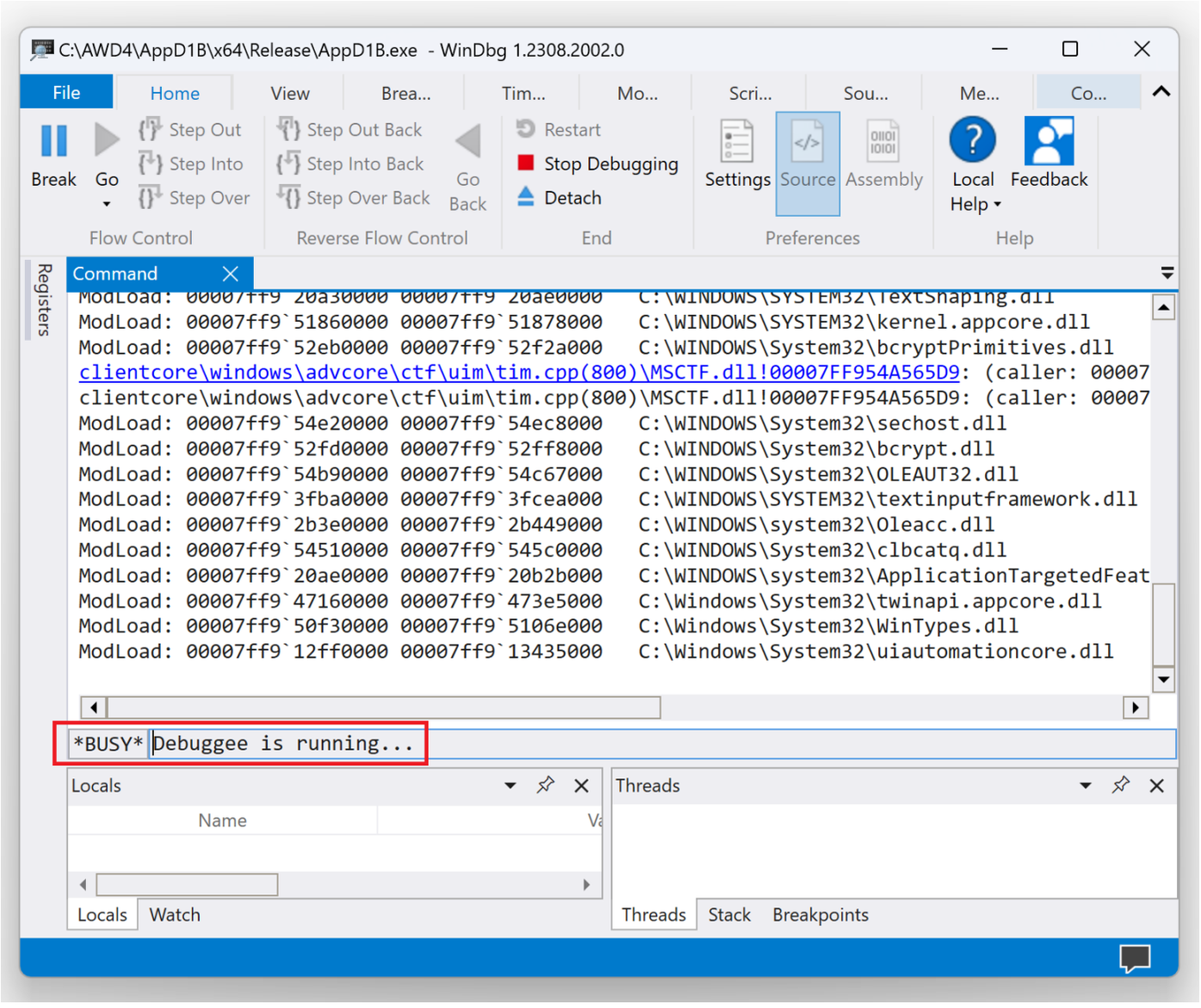
15. Launch another instance of WinDbg and launch *\AWD4\AppD1B\x64\Release\AppD1B.exe* executable. We get the following output:

```
Microsoft (R) Windows Debugger Version 10.0.25921.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: C:\AWD4\AppD1B\x64\Release\AppD1B.exe

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
ModLoad: 00007ff6`bcc80000 00007ff6`bcca0000   AppD1B.exe
ModLoad: 00007ff9`55530000 00007ff9`55747000   ntdll.dll
ModLoad: 00007ff9`55030000 00007ff9`550f4000   C:\WINDOWS\System32\KERNEL32.DLL
ModLoad: 00007ff9`52870000 00007ff9`52c16000   C:\WINDOWS\System32\KERNELBASE.dll
ModLoad: 00007ff9`54130000 00007ff9`542de000   C:\WINDOWS\System32\USER32.dll
ModLoad: 00007ff9`53190000 00007ff9`531b6000   C:\WINDOWS\System32\win32u.dll
ModLoad: 00007ff9`542e0000 00007ff9`54309000   C:\WINDOWS\System32\GDI32.dll
ModLoad: 00007ff9`52c20000 00007ff9`52d38000   C:\WINDOWS\System32\gdi32full.dll
ModLoad: 00007ff9`52f30000 00007ff9`52fca000   C:\WINDOWS\System32\msvcp_win.dll
ModLoad: 00007ff9`53000000 00007ff9`53111000   C:\WINDOWS\System32\ucrtbase.dll
(2228.62e4): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007ff9`5560b784 int     3
```

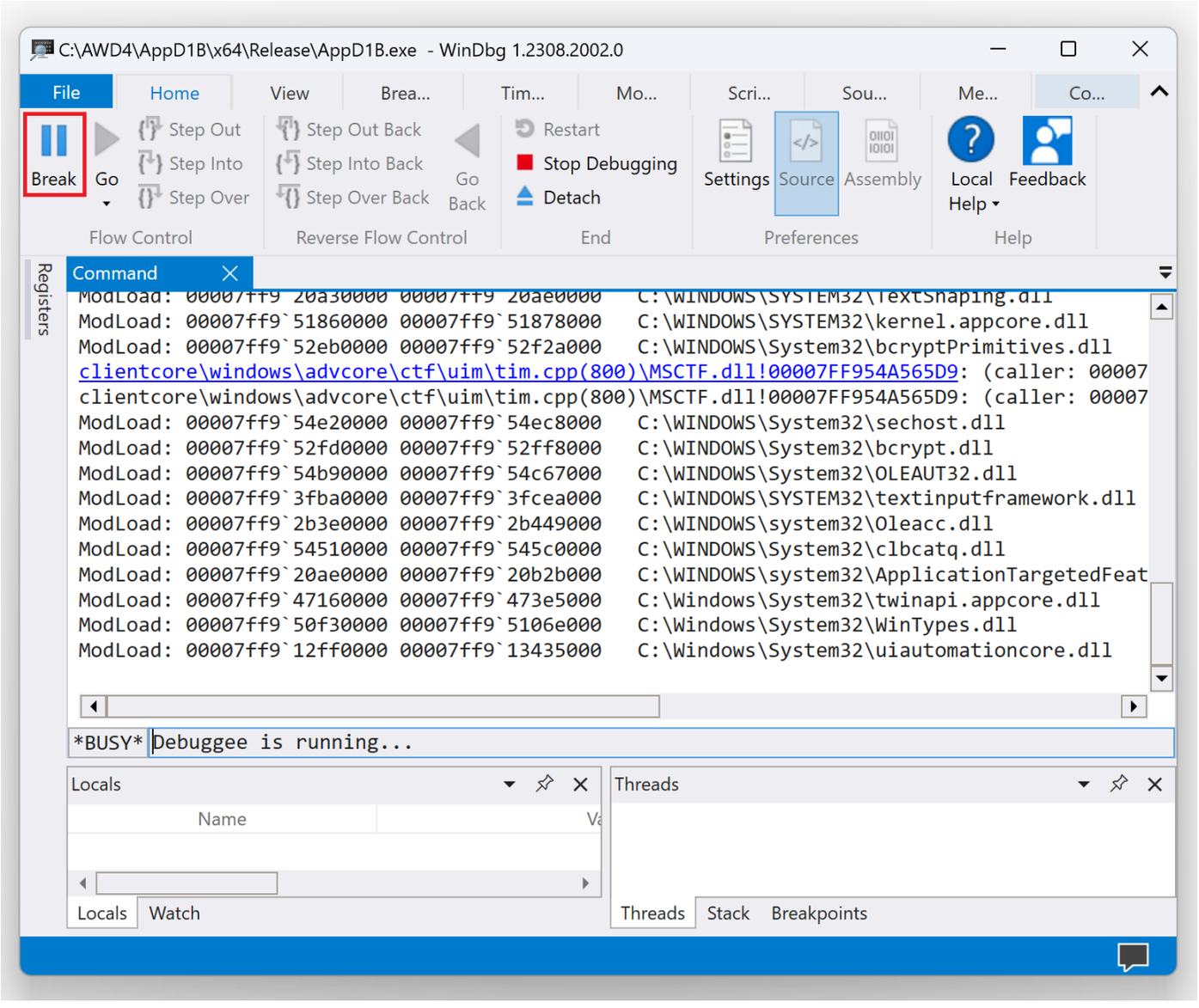
16. We open a new log file:

```
0:000> .logopen C:\AWD4\D1B.log
Opened log file 'C:\AWD4\D1B.log'
```

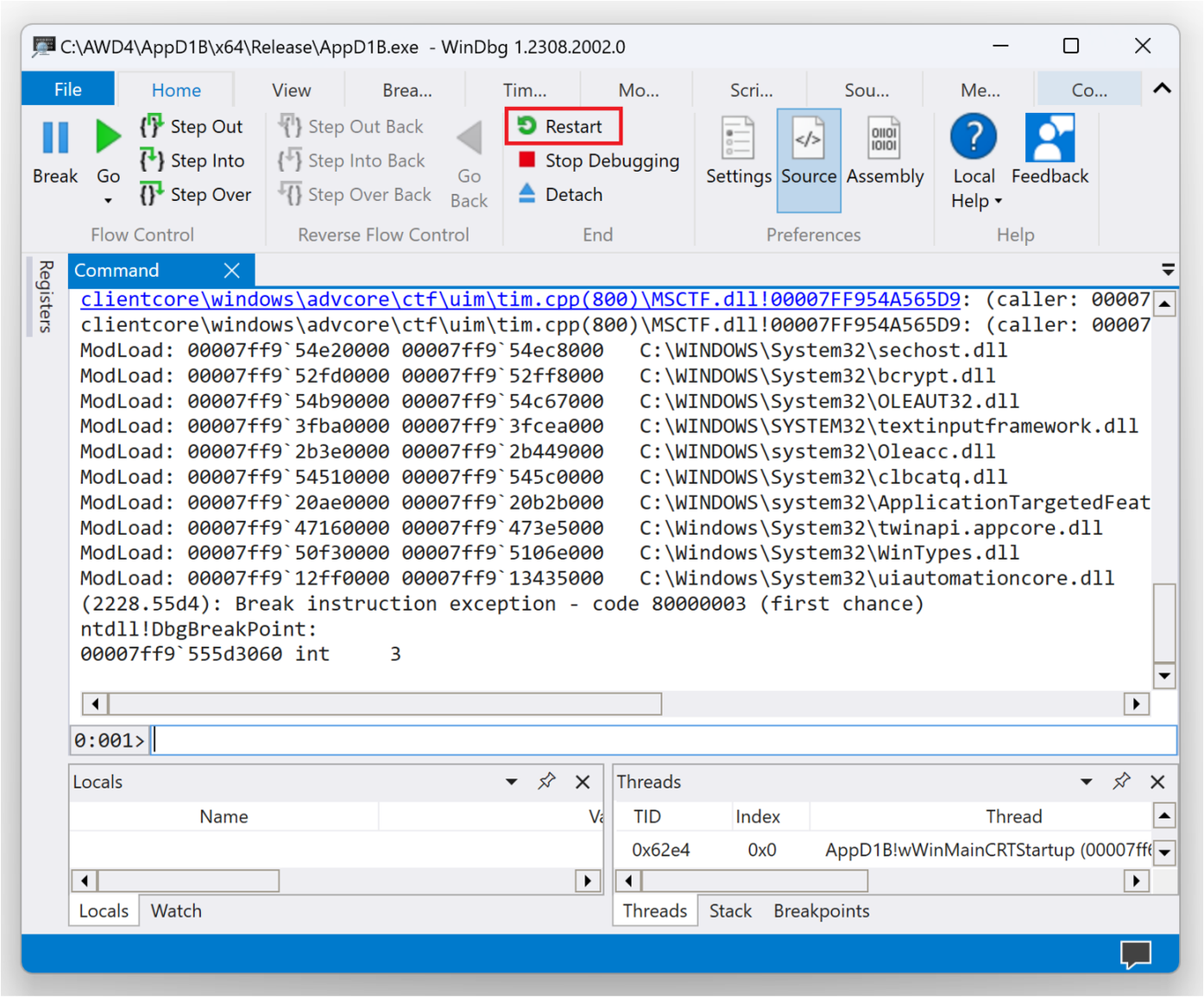
17. If we run it via the **g** command, we don't get any exceptions. We also see that the Debuggee (our program) is running):



18. So we click on the *Home* \ *Break* button:



19. And then on *Home* \ *Restart* (or the **.restart** command):



20. We get the following output:

```
0:000> g
ModLoad: 00007ff9`53ef0000 00007ff9`53f21000   C:\WINDOWS\System32\IMM32.DLL
ModLoad: 00007ff9`4f920000 00007ff9`4f9cb000   C:\WINDOWS\system32\uxtheme.dll
ModLoad: 00007ff9`53b60000 00007ff9`53ee9000   C:\WINDOWS\System32\combase.dll
ModLoad: 00007ff9`543d0000 00007ff9`544e7000   C:\WINDOWS\System32\RPCRT4.dll
ModLoad: 00007ff9`54a40000 00007ff9`54b90000   C:\WINDOWS\System32\MSCTF.dll
ModLoad: 00007ff9`55100000 00007ff9`551a7000   C:\WINDOWS\System32\msvcrt.dll
ModLoad: 00007ff9`20a30000 00007ff9`20ae0000   C:\WINDOWS\SYSTEM32\TextShaping.dll
ModLoad: 00007ff9`51860000 00007ff9`51878000   C:\WINDOWS\SYSTEM32\kernel.appcore.dll
ModLoad: 00007ff9`52eb0000 00007ff9`52f2a000   C:\WINDOWS\System32\bcryptPrimitives.dll
clientcore\windows\advcore\ctf\uim\tim.cpp(800)\MSCTF.dll!00007FF954A565D9: (caller:
00007FF954A5720C) LogHr(1) tid(62e4) 8007029C An assertion failure has occurred.
clientcore\windows\advcore\ctf\uim\tim.cpp(800)\MSCTF.dll!00007FF954A565D9: (caller:
00007FF954A5720C) LogHr(2) tid(62e4) 8007029C An assertion failure has occurred.
ModLoad: 00007ff9`54e20000 00007ff9`54ec8000   C:\WINDOWS\System32\sechost.dll
ModLoad: 00007ff9`52fd0000 00007ff9`52ff8000   C:\WINDOWS\System32\bcrypt.dll
ModLoad: 00007ff9`54b90000 00007ff9`54c67000   C:\WINDOWS\System32\OLEAUT32.dll
```

```
ModLoad: 00007ff9`3fba0000 00007ff9`3fcea000   C:\WINDOWS\SYSTEM32\textinputframework.dll
ModLoad: 00007ff9`2b3e0000 00007ff9`2b449000   C:\WINDOWS\system32\Oleacc.dll
ModLoad: 00007ff9`54510000 00007ff9`545c0000   C:\WINDOWS\System32\clbcatq.dll
ModLoad: 00007ff9`20ae0000 00007ff9`20b2b000
C:\WINDOWS\system32\ApplicationTargetedFeatureDatabase.dll
ModLoad: 00007ff9`47160000 00007ff9`473e5000   C:\Windows\System32\twinapi.appcore.dll
ModLoad: 00007ff9`50f30000 00007ff9`5106e000   C:\Windows\System32\WinTypes.dll
ModLoad: 00007ff9`12ff0000 00007ff9`13435000   C:\Windows\System32\uiautomationcore.dll
(2228.55d4): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ff9`555d3060 int     3

0:001> .restart
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\atlmfc.na
tvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\Objective
C.natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\concurren
cy.natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\cpp_rest.
natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\stl.natvi
s'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\Windows.D
ata.Json.natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\Windows.D
evices.Geolocation.natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\Windows.D
evices.Sensors.natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\Windows.M
edia.natvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\windows.n
atvis'
NatVis script unloaded from 'C:\Program
Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visualizers\winrt.nat
vis'

Microsoft (R) Windows Debugger Version 10.0.25921.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: C:\AWD4\AppD1B\x64\Release\AppD1B.exe

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
ModLoad: 00007ff6`bcc80000 00007ff6`bcca0000   AppD1B.exe
ModLoad: 00007ff9`55530000 00007ff9`55747000   ntdll.dll
ModLoad: 00007ff9`55030000 00007ff9`550f4000   C:\WINDOWS\System32\KERNEL32.DLL
```

```
ModLoad: 00007ff9`52870000 00007ff9`52c16000   C:\WINDOWS\System32\KERNELBASE.dll
ModLoad: 00007ff9`54130000 00007ff9`542de000   C:\WINDOWS\System32\USER32.dll
ModLoad: 00007ff9`53190000 00007ff9`531b6000   C:\WINDOWS\System32\win32u.dll
ModLoad: 00007ff9`542e0000 00007ff9`54309000   C:\WINDOWS\System32\GDI32.dll
ModLoad: 00007ff9`52c20000 00007ff9`52d38000   C:\WINDOWS\System32\gdi32full.dll
ModLoad: 00007ff9`52f30000 00007ff9`52fca000   C:\WINDOWS\System32\msvcp_win.dll
ModLoad: 00007ff9`53000000 00007ff9`53111000   C:\WINDOWS\System32\ucrtbase.dll
(7738.8430): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007ff9`5560b784 int     3
```

21. Since we want to compare the same behavior of the *RegisterClassW* function, we need to put a breakpoint to break in when this function is about to be executed. Then, we would see the *WNDCLASS* structure passed to it. We set a pattern-matching breakpoint using the **bm** command:

```
0:000> bm *!RegisterClassW
  0: 00007ff9`529f6a50 @!"KERNELBASE!RegisterClassW"
  1: 00007ff9`5413f130 @!"USER32!RegisterClassW
```

22. Indeed, we hit the breakpoint immediately:

```
0:000> g
ModLoad: 00007ff9`53ef0000 00007ff9`53f21000   C:\WINDOWS\System32\IMM32.DLL
Breakpoint 1 hit
USER32!RegisterClassW:
00007ff9`5413f130 push    rbx
```

We get an identical stack trace prior to *RegisterClassW* when we compare it with the previously running instance of *AppD1A.exe* (**kL** command omits source code references, **; *** is for comments):

```
0:000> kL; * AppD1B
 # Child-SP          RetAddr               Call Site
00 0000006f`c2defc48 00007ff6`bcc8118d     USER32!RegisterClassW
01 0000006f`c2defc50 00007ff6`bcc8105c     AppD1B!MyRegisterClass+0x8d
02 0000006f`c2defcd0 00007ff6`bcc8168a     AppD1B!wWinMain+0x5c
03 (Inline Function) --------`--------     AppD1B!invoke_main+0x21
04 0000006f`c2defd40 00007ff9`5504257d     AppD1B!__scrt_common_main_seh+0x106
05 0000006f`c2defd80 00007ff9`5558aa58     KERNEL32!BaseThreadInitThunk+0x1d
06 0000006f`c2defdb0 00000000`00000000     ntdll!RtlUserThreadStart+0x28

0:000> kL; * AppD1A
 # Child-SP          RetAddr               Call Site
00 000000e1`7d2ff740 00007ff7`8ac3105c     AppD1A!MyRegisterClass+0x82
01 000000e1`7d2ff7c0 00007ff7`8ac3168a     AppD1A!wWinMain+0x5c
02 (Inline Function) --------`--------     AppD1A!invoke_main+0x21
03 000000e1`7d2ff820 00007ff9`5504257d     AppD1A!__scrt_common_main_seh+0x106
04 000000e1`7d2ff860 00007ff9`5558aa58     KERNEL32!BaseThreadInitThunk+0x1d
05 000000e1`7d2ff890 00000000`00000000     ntdll!RtlUserThreadStart+0x28
```

Note: In the case of an AppD1A crash, we have to execute the **.cxr** command to reset the current stack frame to #0 before executing the **kL** command.

23. We choose frame #1, which called the *RegisterClassW* function, and immediately get access to the *wc* variable (we also note that the function *MyRegisterClass* source code is identical to *AppD1A*):

```
0:000> .frame 1
01 0000006f`c2defc50 00007ff6`bcc8105c     AppD1B!MyRegisterClass+0x8d [C:\AWD4\AppD1B\AppD1B\AppD1B.cpp @ 87]
```

```
0:000> dt wc * AppD1B
Local var @ 0x6fc2defc70 Type tagWNDCLASSW
   +0x000 style  : 3
   +0x008 lpfnWndProc : 0x00007ff6`bcc81260     int64  AppD1B!WndProc+0
   +0x010 cbClsExtra : 0n0
   +0x014 cbWndExtra : 0n0
   +0x018 hInstance : 0x00007ff6`bcc80000 HINSTANCE__
   +0x020 hIcon  : 0x00000000`0425127a HICON__
   +0x028 hCursor : 0x00000000`00010003 HICON__
   +0x030 hbrBackground : 0x00000000`00000006 HBRUSH__
   +0x038 lpszMenuName : 0x00000000`0000006d  "--- memory read error at address 0x00000000`0000006d ---"
   +0x040 lpszClassName : 0x00007ff6`bcc9ad10  "APPD1B"
```

24. But if we look at the AppD1A structure variant, we see its members have different offsets:

```
0:000> dt wc * AppD1A
Local var @ 0xe17d2ff760 Type tagWNDCLASSW
   +0x000 style  : 3
   +0x004 lpfnWndProc : 0x00007ff7`8ac31260     int64  AppD1A!WndProc+0
   +0x00c cbClsExtra : 0n0
   +0x010 cbWndExtra : 0n0
   +0x014 hInstance : 0x00007ff7`8ac30000 HINSTANCE__
   +0x01c hIcon  : 0x00000000`0ea81133 HICON__
   +0x024 hCursor : 0x00000000`00010003 HICON__
   +0x02c hbrBackground : 0x00000000`00000006 HBRUSH__
   +0x034 lpszMenuName : 0x00000000`0000006d  "--- memory read error at address 0x00000000`0000006d ---"
   +0x03c lpszClassName : 0x00007ff7`8ac4ad10  "APPD1A"
```

Note: In the case of an AppD1A crash scenario, we have to switch to frame #4 first.

25. We close logs in both WinDbg instances:

```
0:000> .logclose * AppD1A
Closing open log file C:\AWD4\D1A.log

0:000> .logclose * AppD1B
Closing open log file C:\AWD4\D1B.log
```

Note: We recommend exiting WinDbg after each exercise to avoid possible confusion and glitches.

26. The problem was partially fixed without changing alignment by using a different, bigger *WNDCLASSEX* structure and the *RegisterClassExW* Win32 API function. We launch *\AWD4\AppD1C\x64\Release\AppD1C.exe* in another WinDbg instance:

```
Microsoft (R) Windows Debugger Version 10.0.25921.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: C:\AWD4\AppD1C\x64\Release\AppD1C.exe

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
ModLoad: 00007ff6`d0d30000 00007ff6`d0d50000   AppD1C.exe
ModLoad: 00007ff9`55530000 00007ff9`55747000   ntdll.dll
ModLoad: 00007ff9`55030000 00007ff9`550f4000   C:\WINDOWS\System32\KERNEL32.DLL
ModLoad: 00007ff9`52870000 00007ff9`52c16000   C:\WINDOWS\System32\KERNELBASE.dll
ModLoad: 00007ff9`54130000 00007ff9`542de000   C:\WINDOWS\System32\USER32.dll
ModLoad: 00007ff9`53190000 00007ff9`531b6000   C:\WINDOWS\System32\win32u.dll
```

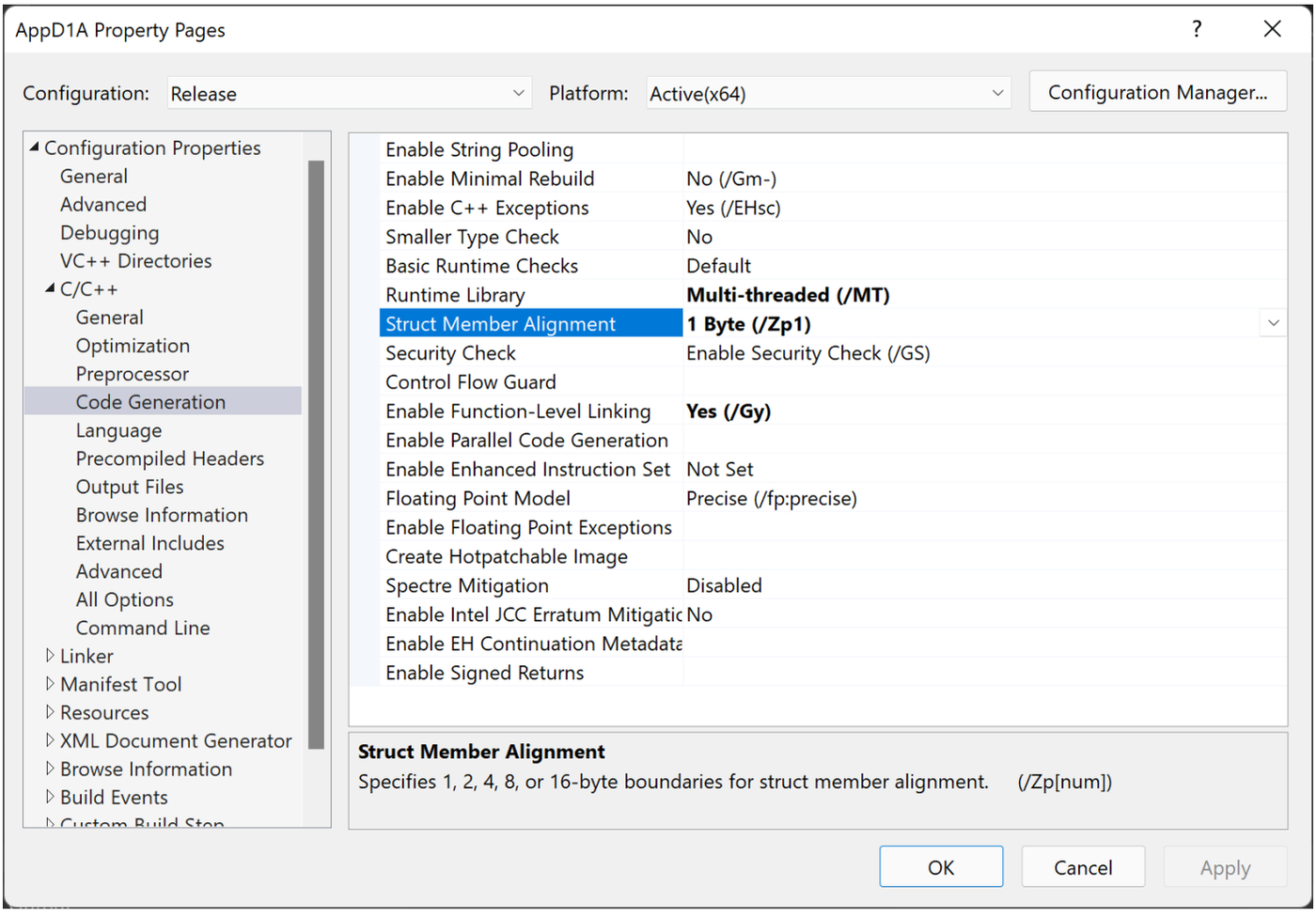
```
ModLoad: 00007ff9`542e0000 00007ff9`54309000   C:\WINDOWS\System32\GDI32.dll
ModLoad: 00007ff9`52c20000 00007ff9`52d38000   C:\WINDOWS\System32\gdi32full.dll
ModLoad: 00007ff9`52f30000 00007ff9`52fca000   C:\WINDOWS\System32\msvcp_win.dll
ModLoad: 00007ff9`53000000 00007ff9`53111000   C:\WINDOWS\System32\ucrtbase.dll
(8b48.7758): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007ff9`5560b784 int     3

0:000> bm *!RegisterClassExW
  0: 00007ff9`529f6a20 @!"KERNELBASE!RegisterClassExW"
  1: 00007ff9`5413ef30 @!"USER32!RegisterClassExW"

0:000> g
ModLoad: 00007ff9`53ef0000 00007ff9`53f21000   C:\WINDOWS\System32\IMM32.DLL
Breakpoint 1 hit
USER32!RegisterClassExW:
00007ff9`5413ef30 sub     rsp,38h

0:000> kL
 # Child-SP          RetAddr               Call Site
00 00000052`c21ef7e8 00007ff6`d0d311aa     USER32!RegisterClassExW
01 00000052`c21ef7f0 00007ff6`d0d3105c     AppD1C!MyRegisterClass+0xaa
02 00000052`c21ef870 00007ff6`d0d3169a     AppD1C!wWinMain+0x5c
03 (Inline Function) --------`--------     AppD1C!invoke_main+0x21
04 00000052`c21ef8d0 00007ff9`5504257d     AppD1C!__scrt_common_main_seh+0x106
05 00000052`c21ef910 00007ff9`5558aa58     KERNEL32!BaseThreadInitThunk+0x1d
06 00000052`c21ef940 00000000`00000000     ntdll!RtlUserThreadStart+0x28

0:000> .frame 1
01 00000052`c21ef7f0 00007ff6`d0d3105c     AppD1C!MyRegisterClass+0xaa [C:\AWD4\AppD1C\AppD1C\AppD1C.cpp @ 87]

0:000> dv /i /V
prv param  00000052`c21ef870 @rsp+0x0080             hInstance = 0x00007ff6`d0d30000
prv local  00000052`c21ef810 @rsp+0x0020                  wcex = struct tagWNDCLASSEXW
```

Note: Adding a new extra member in the new structure shifts the remaining members and sets the same layout as in *AppD1B*:

```
0:000> dt wcex * AppD1C
Local var @ 0x52c21ef810 Type tagWNDCLASSEXW
   +0x000 cbSize : 0x50
   +0x004 style  : 3
   +0x008 lpfnWndProc : 0x00007ff6`d0d31270     int64  AppD1C!WndProc+0
   +0x010 cbClsExtra : 0n0
   +0x014 cbWndExtra : 0n0
   +0x018 hInstance : 0x00007ff6`d0d30000 HINSTANCE__
   +0x020 hIcon  : 0x00000000`00b11998 HICON__
   +0x028 hCursor : 0x00000000`00010003 HICON__
   +0x030 hbrBackground : 0x00000000`00000006 HBRUSH__
   +0x038 lpszMenuName : 0x00000000`0000006d  "--- memory read error at address 0x00000000`0000006d ---"
   +0x040 lpszClassName : 0x00007ff6`d0d4ad10  "APPD1C"
   +0x048 hIconSm : 0x00000000`00ec19f2 HICON__
```

```
0:000> dt wc * AppD1B
Local var @ 0x6fc2defc70 Type tagWNDCLASSW
   +0x000 style  : 3
   +0x008 lpfnWndProc : 0x00007ff6`bcc81260     int64  AppD1B!WndProc+0
   +0x010 cbClsExtra : 0n0
   +0x014 cbWndExtra : 0n0
   +0x018 hInstance : 0x00007ff6`bcc80000 HINSTANCE__
   +0x020 hIcon  : 0x00000000`0425127a HICON__
   +0x028 hCursor : 0x00000000`00010003 HICON__
   +0x030 hbrBackground : 0x00000000`00000006 HBRUSH__
   +0x038 lpszMenuName : 0x00000000`0000006d  "--- memory read error at address 0x00000000`0000006d ---"
   +0x040 lpszClassName : 0x00007ff6`bcc9ad10  "APPD1B"
```

Note: *AppD1A* wasn't working because of structure member alignment. This models an old Windows 3.x project that was ported to x64. It had the minimum alignment in the past to reduce memory consumption:



Note: *AppD1B* was working because the alignment was changed to default. *AppD1C* still used the same 1-byte alignment, but because the bigger structure shifted members of the substructure, it didn't fail the Windows API call (or didn't crash).