Debugging Windows Applications with IDA WinDbg Plugin

Table of Contents

Last updated on August 18, 2020 — v0.1

1. Setup

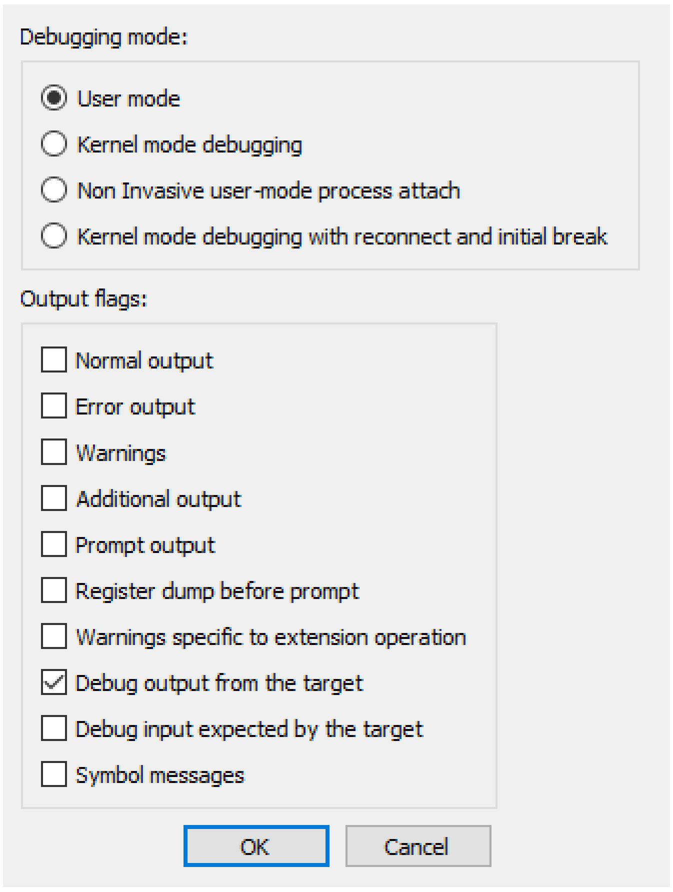
The Windbg debugger plugin is an IDA Pro debugger plugin that uses Microsoft's debugging engine (dbgeng) that is used by Windbg, Cdb or Kd.

To get started, you need to install the latest Debugging Tools from Microsoft website: Download the Windows Driver Kit (WDK) or from the Windows SDK / DDK package.

Please use **ar.exe.idb** from samples.zip to follow this tutorial.

After installing the debugging tools, make sure you select "Debugger > Switch Debugger" and select the WinDbg debugger.

Also make sure you specify the correct settings in the "Debugger specific options" dialog:



Debugging mode:

- [✓ User mode]
- Kernel mode debugging
- Non Invasive user-mode process attach
- Kernel mode debugging with reconnect and initial break

Output flags:

- Normal output
- Error output
- Warnings
- Additional output
- Prompt output
- Register dump before prompt
- Warnings specific to extension operation
- [✓ Debug output from the target]
- Debug input expected by the target
- Symbol messages

User mode

Select this mode for user mode application debugging (default mode)

Kernel mode

Select this mode to attach to a live kernel.

Non Invasive debugging

Select this mode to attach to a process non-invasively

Output flags

These flags tell the debugging engine which kind of output messages to display and which to omit

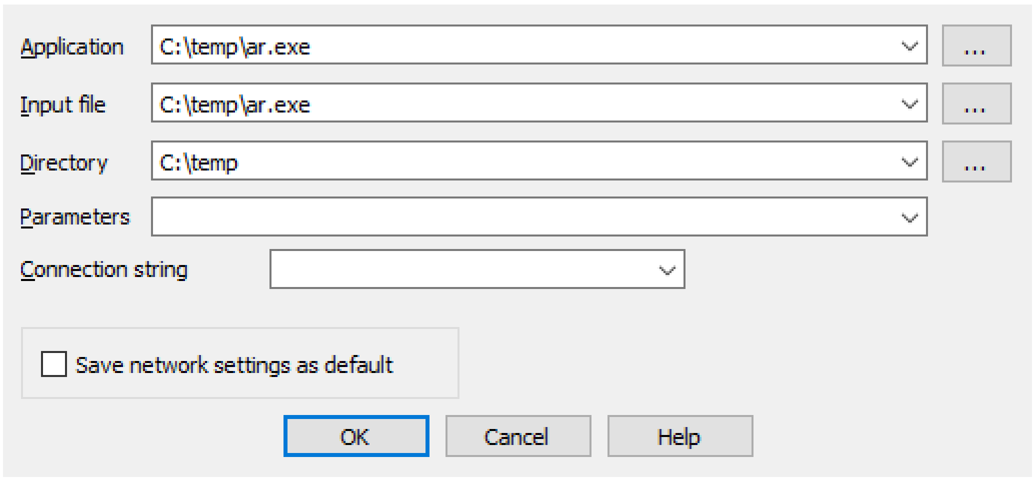
Kernel mode debugging with reconnect and initial break

Select this option when debugging a kernel and when the connection string contains 'reconnect'. This option will assure that the debugger breaks as soon as possible after a reconnect.

To make these settings permanent, please edit the `%IDA%\cfg\dbg_windbg.cfg` file.

To specify the debugging tools folder you may add to the PATH environment variable the location of Windbg.exe or edit `%IDA%\cfg\ida.cfg` and change the value of the DBGTOOLS key.

After the debugger is properly configured, edit the process options and leave the connection string value empty because we intend to debug a local user-mode application.

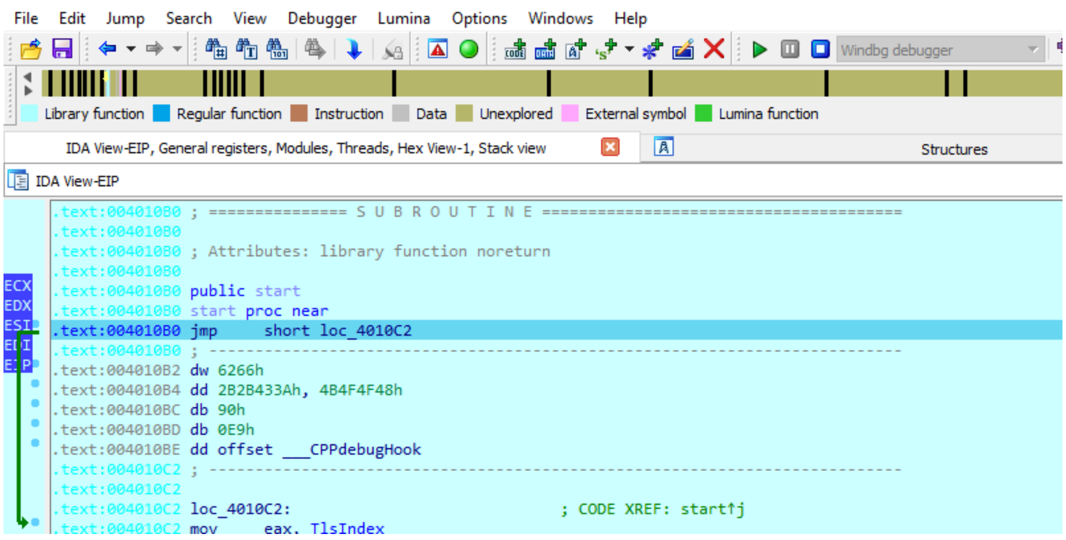


Application	C:\temp\ar.exe
Input file	C:\temp\ar.exe
Directory	C:\temp
Parameters	
Connection string	

☐ Save network settings as default

2. Starting the debugger

Now hit F9 to start debugging:

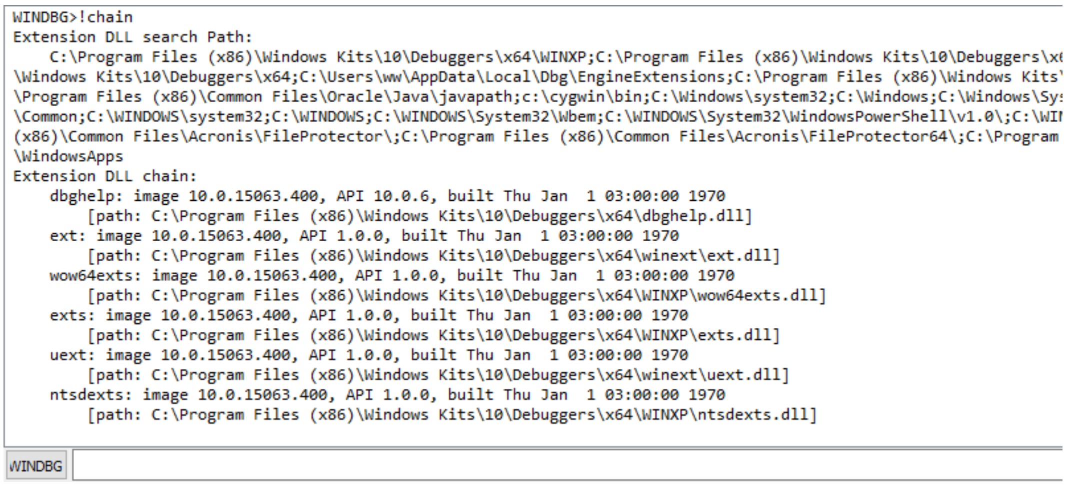


File Edit Jump Search View Debugger Lumina Options Windows Help
Windbg debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
IDA View-EIP, General registers, Modules, Threads, Hex View-1, Stack view
Structures
IDA View-EIP
ECX
EDX
ESI
EDI
EIP

```
.text:004010B0 ; =============== S U B R O U T I N E =======================================
.text:004010B0
.text:004010B0 ; Attributes: library function noreturn
.text:004010B0
.text:004010B0 public start
.text:004010B0 start proc near
.text:004010B0 jmp     short loc_4010C2
.text:004010B0 ; ---------------------------------------------------------------------------
.text:004010B2 dw 6266h
.text:004010B4 dd 2B2B433Ah, 4B4F4F48h
.text:004010BC db 90h
.text:004010BD db 0E9h
.text:004010BE dd offset ___CPPdebugHook
.text:004010C2 ; ---------------------------------------------------------------------------
.text:004010C2
.text:004010C2 loc_4010C2:                             ; CODE XREF: start↑j
.text:004010C2 mov     eax, TlsIndex
```

The Windbg plugin is very similar to IDA Pro's Win32 debugger plugin, nonetheless by using the former, one can benefit from the command line facilities and the extensions that ship with the debugging tools.

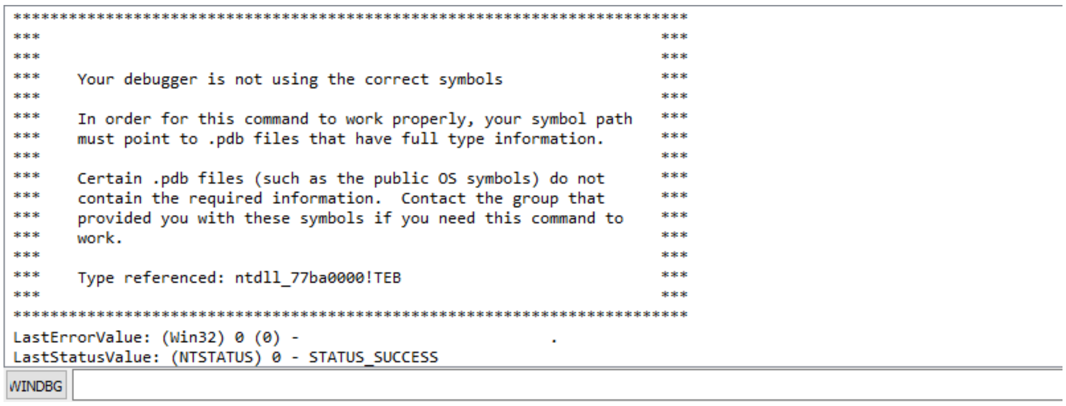
For example, one can type `!chain` to see the registered Windbg extensions:



```
WINDBG>!chain
Extension DLL search Path:
    C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\WINXP;C:\Program Files (x86)\Windows Kits\10\Debuggers\x
\Windows Kits\10\Debuggers\x64;C:\Users\ww\AppData\Local\Dbg\EngineExtensions;C:\Program Files (x86)\Windows Kits
\Program Files (x86)\Common Files\Oracle\Java\javapath;c:\cygwin\bin;C:\Windows\system32;C:\Windows;C:\Windows\Sy
\Common;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WI
(x86)\Common Files\Acronis\FileProtector\;C:\Program Files (x86)\Common Files\Acronis\FileProtector64\;C:\Program
\WindowsApps
Extension DLL chain:
    dbghelp: image 10.0.15063.400, API 10.0.6, built Thu Jan  1 03:00:00 1970
        [path: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\dbghelp.dll]
    ext: image 10.0.15063.400, API 1.0.0, built Thu Jan  1 03:00:00 1970
        [path: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\winext\ext.dll]
    wow64exts: image 10.0.15063.400, API 1.0.0, built Thu Jan  1 03:00:00 1970
        [path: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\WINXP\wow64exts.dll]
    exts: image 10.0.15063.400, API 1.0.0, built Thu Jan  1 03:00:00 1970
        [path: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\WINXP\exts.dll]
    uext: image 10.0.15063.400, API 1.0.0, built Thu Jan  1 03:00:00 1970
        [path: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\winext\uext.dll]
    ntsdexts: image 10.0.15063.400, API 1.0.0, built Thu Jan  1 03:00:00 1970
        [path: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\WINXP\ntsdexts.dll]
```

WINDBG

`!gle` is another command to get the last error value of a given Win32 API call.



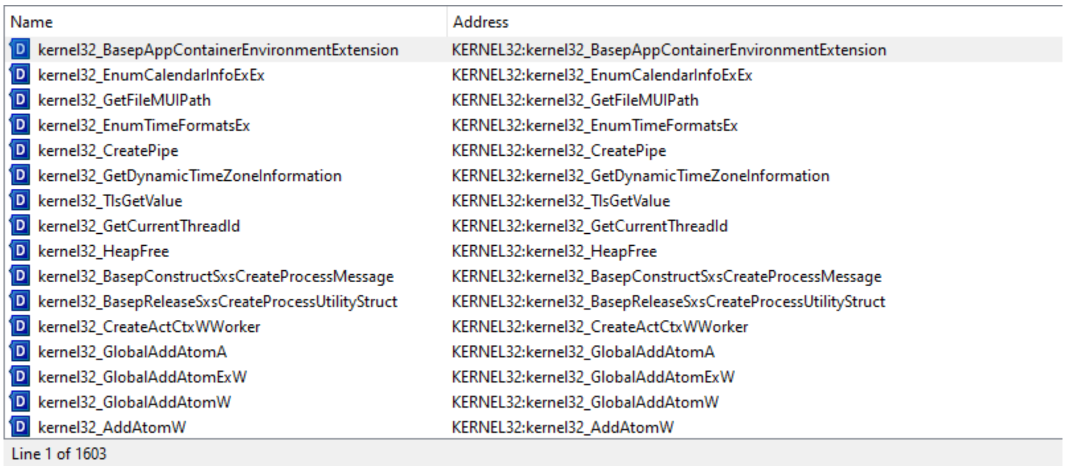
```
*************************************************************************
***                                                                   ***
***                                                                   ***
***    Your debugger is not using the correct symbols                 ***
***                                                                   ***
***    In order for this command to work properly, your symbol path   ***
***    must point to .pdb files that have full type information.      ***
***                                                                   ***
***    Certain .pdb files (such as the public OS symbols) do not      ***
***    contain the required information.  Contact the group that      ***
***    provided you with these symbols if you need this command to    ***
***    work.                                                          ***
***                                                                   ***
***    Type referenced: ntdll_77ba0000!TEB                            ***
***                                                                   ***
*************************************************************************
LastErrorValue: (Win32) 0 (0) -                                 .
LastStatusValue: (NTSTATUS) 0 - STATUS_SUCCESS
```

WINDBG

3. Use of symbolic information

Another benefit of using the Windbg debugger plugin is the use of symbolic information.

Normally, if the debugging symbols path is not set, then the module window will only show the exported names. For example kernel32.dll displays 1603 names:[1]



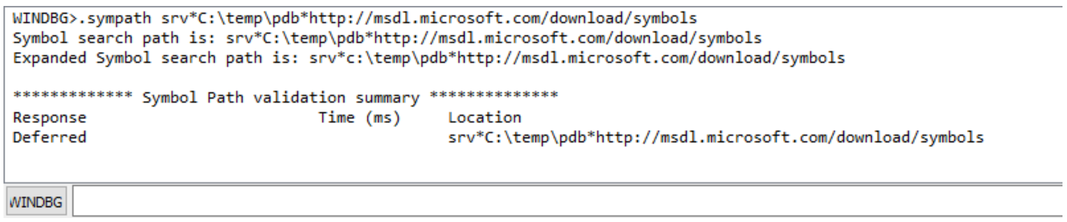
Name	Address
kernel32_BasepAppContainerEnvironmentExtension	KERNEL32:kernel32_BasepAppContainerEnvironmentExtension
kernel32_EnumCalendarInfoExEx	KERNEL32:kernel32_EnumCalendarInfoExEx
kernel32_GetFileMUIPath	KERNEL32:kernel32_GetFileMUIPath
kernel32_EnumTimeFormatsEx	KERNEL32:kernel32_EnumTimeFormatsEx
kernel32_CreatePipe	KERNEL32:kernel32_CreatePipe
kernel32_GetDynamicTimeZoneInformation	KERNEL32:kernel32_GetDynamicTimeZoneInformation
kernel32_TlsGetValue	KERNEL32:kernel32_TlsGetValue
kernel32_GetCurrentThreadId	KERNEL32:kernel32_GetCurrentThreadId
kernel32_HeapFree	KERNEL32:kernel32_HeapFree
kernel32_BaseConstructSxsCreateProcessMessage	KERNEL32:kernel32_BaseConstructSxsCreateProcessMessage
kernel32_BasepReleaseSxsCreateProcessUtilityStruct	KERNEL32:kernel32_BasepReleaseSxsCreateProcessUtilityStruct
kernel32_CreateActCtxWWorker	KERNEL32:kernel32_CreateActCtxWWorker
kernel32_GlobalAddAtomA	KERNEL32:kernel32_GlobalAddAtomA
kernel32_GlobalAddAtomExW	KERNEL32:kernel32_GlobalAddAtomExW
kernel32_GlobalAddAtomW	KERNEL32:kernel32_GlobalAddAtomW
kernel32_AddAtomW	KERNEL32:kernel32_AddAtomW

Line 1 of 1603

Let us configure a symbol source by adding this environment variable before running IDA:

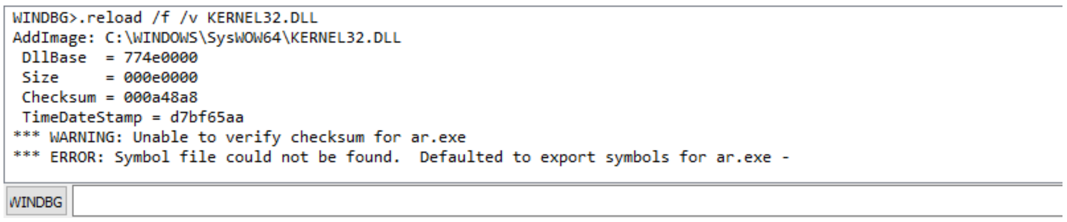
```
set _NT_SYMBOL_PATH=srv*C:\temp\pdb*http://msdl.microsoft.com/download/symbols
```

It is also possible to set the symbol path directly while debugging typing `.sympath srv*C:\temp\pdb*http://msdl.microsoft.com/download/symbols` in the WINDBG console:



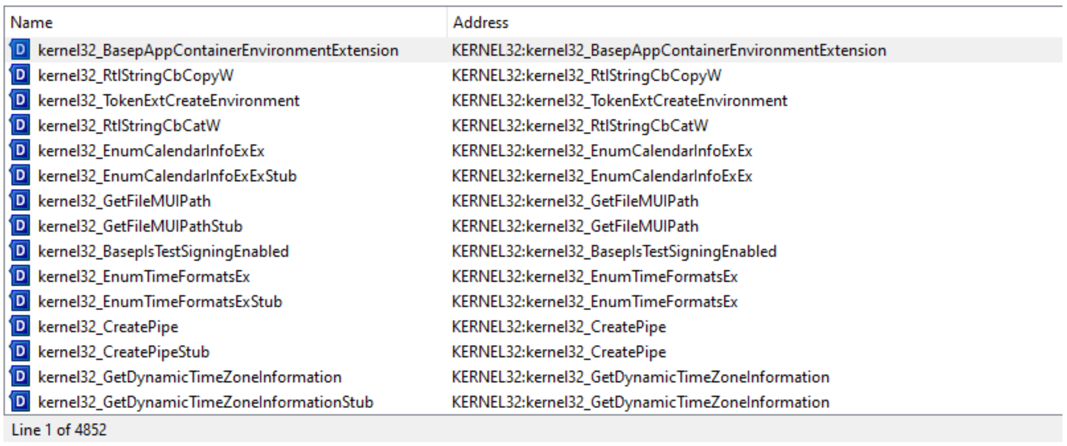
```
WINDBG>.sympath srv*C:\temp\pdb*http://msdl.microsoft.com/download/symbols
Symbol search path is: srv*C:\temp\pdb*http://msdl.microsoft.com/download/symbols
Expanded Symbol search path is: srv*c:\temp\pdb*http://msdl.microsoft.com/download/symbols

************ Symbol Path validation summary *************
Response                         Time (ms)     Location
Deferred                                       srv*C:\temp\pdb*http://msdl.microsoft.com/download/symbols
```

and then typing `.reload /f /v KERNEL32.DLL` to reload the symbols:



```
WINDBG>.reload /f /v KERNEL32.DLL
AddImage: C:\WINDOWS\SysWOW64\KERNEL32.DLL
 DllBase  = 774e0000
 Size     = 000e0000
 Checksum = 000a48a8
 TimeDateStamp = d7bf65aa
*** WARNING: Unable to verify checksum for ar.exe
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ar.exe -
```

Now we try again and notice that more symbol names are retrieved from kernel32.dll:

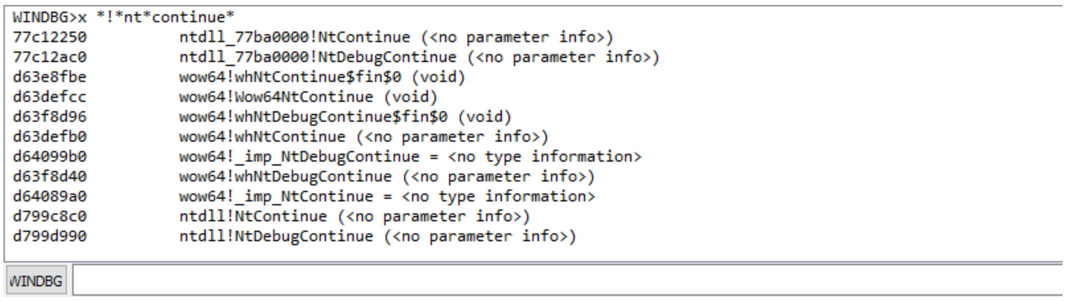


Name	Address
kernel32_BaseppAppContainerEnvironmentExtension	KERNEL32:kernel32_BasepAppContainerEnvironmentExtension
kernel32_RtlStringCbCopyW	KERNEL32:kernel32_RtlStringCbCopyW
kernel32_TokenExtCreateEnvironment	KERNEL32:kernel32_TokenExtCreateEnvironment
kernel32_RtlStringCbCatW	KERNEL32:kernel32_RtlStringCbCatW
kernel32_EnumCalendarInfoExEx	KERNEL32:kernel32_EnumCalendarInfoExEx
kernel32_EnumCalendarInfoExExStub	KERNEL32:kernel32_EnumCalendarInfoExEx
kernel32_GetFileMUIPath	KERNEL32:kernel32_GetFileMUIPath
kernel32_GetFileMUIPathStub	KERNEL32:kernel32_GetFileMUIPath
kernel32_BaseplsTestSigningEnabled	KERNEL32:kernel32_BaseplsTestSigningEnabled
kernel32_EnumTimeFormatsEx	KERNEL32:kernel32_EnumTimeFormatsEx
kernel32_EnumTimeFormatsExStub	KERNEL32:kernel32_EnumTimeFormatsEx
kernel32_CreatePipe	KERNEL32:kernel32_CreatePipe
kernel32_CreatePipeStub	KERNEL32:kernel32_CreatePipe
kernel32_GetDynamicTimeZoneInformation	KERNEL32:kernel32_GetDynamicTimeZoneInformation
kernel32_GetDynamicTimeZoneInformationStub	KERNEL32:kernel32_GetDynamicTimeZoneInformation

Line 1 of 4852

Now we have 4852 symbols instead!

It is also possible to use the `x *!*nt*continue` [2] command to quickly search for symbols:



```
WINDBG>x *!*nt*continue*
77c12250          ntdll_77ba0000!NtContinue (<no parameter info>)
77c12ac0          ntdll_77ba0000!NtDebugContinue (<no parameter info>)
d63e8fbe          wow64!whNtContinue$fin$0 (void)
d63defcc          wow64!Wow64NtContinue (void)
d63f8d96          wow64!whNtDebugContinue$fin$0 (void)
d63defb0          wow64!whNtContinue (<no parameter info>)
d64099b0          wow64!_imp_NtDebugContinue = <no type information>
d63f8d40          wow64!whNtDebugContinue (<no parameter info>)
d64089a0          wow64!_imp_NtContinue = <no type information>
d799c8c0          ntdll!NtContinue (<no parameter info>)
d799d990          ntdll!NtDebugContinue (<no parameter info>)
```

4. Debugging a remote process

We have seen how to debug a local user mode program, now let us see how to debug a remote process. First let us assume that `pcA` is the target machine (where we will run the debugger server and the debugged program) and `pcB` is the machine where IDA Pro and the debugging tools are installed.

To start a remote process:

- On `pcA`, type: `dbgsrv -t tcp:port=5000` [3]
- On `pcB`, setup IDA Pro and Windbg debugger plugin:
  - **Application/Input file**: these should contain a path to the debuggee residing in `pcA`
  - **Connection string**: `tcp:port=5000,server=pcA`

Now run the program and debug it remotely.

To attach to a remote process, use the same steps to setup `pcA` and use the same connection string when attaching to the process.

More about connection strings and different protocols (other than TCP/IP) can be found in `debugger.chm` in the debugging tools folder.

[1] Double click at `KERNEL32.DLL` in `Modules` window to see this list.

[2] Looking for any symbol in any module that contains the word 'continue' after 'nt'

[3] change the port number as needed