|  |  |  |
| --- | --- | --- |
| **1) Built-in help commands** | | |
| **Cmd** | **Variants / Params** | **Description** |
| ? | ?  ? /D | Display regular commands  Display regular commands as DML |
| .help | .help  .help /D  .help /D a\* | Display . commands  Display . commands in DML format (top bar of links is given)  Display . commands that start with a\* (wildcard) as DML |
| .chain | .chain  .chain /D | Lists all loaded debugger extensions  Lists all loaded debugger extensions as DML (where extensions are linked to a .extmatch) |
| .extmatch | .extmatch /e ExtDLL FunctionFilter  .extmatch /D /e ExtDLL FunctionFilter | Show all exported functions of an extension DLL. *FunctionFilter* = wildcard string  Same in DML format (functions link to "!ExtName.help FuncName" commands)   Example: **.extmatch /D /e uext \*** (show all exported functions of uext.dll) |
| .hh | .hh  .hh Text | Open WinDbg's help  Text = text to look up in the help file index  Example: **.hh dt** |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **2) General WinDbg's commands (show version, clear screen, etc.)** | | |
| **Cmd** | **Variants / Params** | **Description** |
| version |  | Dump version info of debugger and loaded extension DLLs |
| vercommand |  | Dump command line that was used to start the debugger |
| vertarget |  | Version of target computer |
| CTRL+ALT+V |  | Toggle verbose mode ON/OFF  In verbose mode some commands (such as register dumping) have more detailed output. |
| n | n [8 | 10 | 16] | Set number base |
| .formats | .formats Expression | Show number formats = evaluates a numerical expression or symbol and displays it in multiple numerical formats (hex, decimal, octal, binary, time, ..)  Example 1: .formats 5  Example 2: .formats poi(nLocal1) == .formats @@($!nLocal1) |
| .cls |  | Clear screen |
| .lastevent |  | Displays the most recent exception or event that occurred (why the debugger is waiting?) |
| .effmach | .effmach  .effmach .  .effmach #  .effmach x86 | amd64 | ia64 | ebc | Dump effective machine (x86, amd64, ..):  Use target computer's native processor mode  Use processor mode of the code that is executing for the most recent event  Use x86, amd64, ia64, or ebc processor mode   This setting influences many debugger features:  -> which processor's unwinder is used for stack tracing  -> which processor's register set is active |
| .time |  | display time (system-up, process-up, kernel time, user time) |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **3) Debugging sessions (attach, detach, ..)** | | |
| **Cmd** | **Variants / Params** | **Description** |
| .attach | PID | attach to a process |
| .detach |  | ends the debugging session, but leaves any user-mode target application running |
| q | q, qq | Quit = ends the debugging session and terminates the target application  Remote debugging: q= no effect; qq= terminates the debug server |
| .restart |  | Restart target application |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **4) Expressions and commands** | | |
| **Cmd** | **Variants / Params** | **Description** |
| ; |  | Command separator (cm1; cm2; ..) |
| ? | ? Expression  ?? Expression | Evaluate expression (use default evaluator)  Evaluate c++ expression |
| .expr | .expr  .expr /q  .expr /s c++  .expr /s masm | Choose default expression evaluator  Show current evaluator  Show available evaluators  Set **c++** as the default expression evaluator  Set **masm** as the default expression evaluator |
| \* | \* [any text] | Comment Line Specifier  Terminated by: end of line |
| $$ | $$ [any text] | Comment Specifier  Terminated by: end of line OR semicolon |
| .echo | .echo String  .echo "String" | Echo Comment -> comment text + echo it  Terminated by: end of line OR semicolon  With the $$ token or the \* token the debugger will ignore the inputted text without echoing it. |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **5) Debugger markup language (DML)   Starting with the 6.6.07 version of the debugger a new mechanism for enhancing output from the debugger and extensions was included: DML.  DML allows output to include directives and extra non-display information in the form of tags.  Debugger user interfaces parse out the extra information to provide new behaviors.   DML is primarily intended to address two issues:**   * **Linking of related information** * **Discoverability of debugger and extension functionality** | | |
| **Cmd** | **Variants / Params** | **Description** |
| .dml\_start |  | Kick of to other DML commands |
| .prefer\_dml | .prefer\_dml [1 | 0] | Global setting: should DML-enhanced commands default to DML?  Note that many commands like k, lm, .. output DML content thereafter. |
| .help /D |  | .help has a new DML mode where a top bar of links is given |
| .chain /D |  | .chain has a new DML mode where extensions are linked to a .extmatch |
| .extmatch /D |  | .extmatch has a new DML format where exported functions link to "!ExtName.help FuncName" commands |
| lmD |  | lm has a new DML mode where module names link to lmv commands |
| kM |  | k has a new DML mode where frame numbers link to a .frame/dv |
| .dml\_flow | .dml\_flow StartAddr TargetAddr | Allows for interactive exploration of code flow for a function.   1. Builds a code flow graph for the function starting at the given start address (similar to uf) 2. Shows the basic block given the target address plus links to referring blocks and blocks referred to by the current block   Example: **.dml\_flow CreateRemoteThread CreateRemoteThread+30** |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **6) Main extensions** | | |
| **Cmd** | **Variants / Params** | **Display supported commands for ..** |
| !Ext.help |  | General extensions |
| !Exts.help |  | -||- |
| !Uext.help |  | User-Mode Extensions (non-OS specific) |
| !Ntsdexts.help |  | User-Mode Extensions (OS specific) |
| !logexts.help |  | Logger Extensions |
| !clr10\sos.help |  | Debugging managed code |
| !wow64exts.help |  | Wow64 debugger extensions |
| !Wdfkd.help |  | Kernel-Mode driver framework extensions |
| !Gdikdx.help |  | Graphics driver extensions |
| .. |  |  |
| !NAME.help | !NAME.help FUNCTION | Display detailed help about an exported function  NAME = placeholder for extension DLL  FUNCTION = placeholder for exported function   Example: **!Ntsdexts.help handle** (show detailed help about !Ntsdexts.handle) |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **7) Symbols** | | |
| **Cmd** | **Variants / Params** | **Description** |
| ld | ld ModuleName  ld \* | Load symbols for Module  Load symbols for all modules |
| !sym | !sym  !sym noisy  !sym quiet | Get state of symbol loading  Set **noisy** symbol loading (debugger displays info about its search for symbols)  Set **quiet** symbol loading (=default) |
| x | x *[Options]* Module!Symbol  x */t* ..  x */v* ..  x */a* ..  x */n* ..  x */z* .. | **Examine symbols**: displays symbols that match the specified pattern  with data type  verbose (symbol type and size)  sort by address  sort by name  sort by size ("size" of a function symbol is the size of the function in memory) |
| ln | ln Addr | **List nearest symbols** = display the symbols at or near the given Addr. Useful to:   * determine what a pointer is pointing to * when looking at a corrupted stack to determine which procedure made a call |
| .sympath | .sympath  .sympath+ | Display or set symbol search path  Append directories to previous symbol path |
| .symopt | .symopt  .symopt+ *Flags*  .symopt- *Flags* | displays current symbol options  add option  remove option |
| .symfix | .symfix  .symfix+ DownstreamStore | Set symbol store path to automatically point to http://msdl.microsoft.com/download/symbols  + = append it to the existing path  DownstreamStore = directory to be used as a downstream store. Default is WinDbgInstallationDir\Sym. |
| .reload | .reload  .reload [/f | /v]  .reload [/f | /v] Module | Reload symbol information for all modules\*\*  f = force immediate symbol load (overrides lazy loading); v = verbose mode  Module = for Module only   \*\*Note: The .reload command does not actually cause symbol information to be read. It just lets the debugger know that the symbol files may have changed, or that a new module should be added to the module list. To force actual symbol loading to occur use the /f option, or the ld (Load Symbols) command. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | x \*! | list all modules | | x ntdll!\* | list all symbols of ntdll | | x /t /v MyDll!\* | list all symbol in MyDll with data type, symbol type and size | | x kernel32!\*LoadLib\* | list all symbols in kernel32 that contain the word LoadLib | |  | | | .sympath+ C:\MoreSymbols | add symbols from C:\MoreSymbols (folder location) | |  | | | .reload /f @"ntdll.dll" | Immediately reload symbols for ntdll.dll. | | .reload /f @"C:\WINNT\System32\verifier.dll" | Reload symbols for verifier. Use the given path. |   Also check the "!lmi" command. |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **8) Sources** | | |
| **Cmd** | **Variants / Params** | **Description** |
| .srcpath | .srcpath  .srcpath+ DIR | Display or set source search path  Append directory to the searched source path |
| .srcnoisy | *{1|0}* | Controls noisy source loading |
| .lines | [-e | -d | -t] | Toggle source line support: enable; disable; toggle |
| l (small letter L) | l+l, l-l  l+o, l-o  l+s, l-s  l+t, l-t | show line numbers  suppress all but [s]  source and line number  source mode vs. assembly mode |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **9) Exceptions, events, and crash analysis** | | |
| **Cmd** | **Variants / Params** | **Description** |
| g | g  gH  gN | Go  Go exception handled  Go not handled |
| .lastevent |  | What happened? Shows most recent event or exception |
| !analyze | !analyze -v  !analyze -hang  !analyze -f | Display information about the current exception or bug check; verbose  User mode: Analyzes the thread stack to determine whether any threads are blocking other threads.  See an exception analysis even when the debugger does not detect an exception. |
| sx | sx  sxe  sxd  sxn  sxi  sxr | Show all event filters with break status and handling  break first-chance  break second-chance  notify; don't break  ignore event  reset filter settings to default values |
| .exr | .exr-1  .exr Addr | display most recent exception record  display exception record at Addr |
| .ecxr |  | displays exception context record (registers) associated with the current exception |
| !cppexr | Addr | Display content and type of C++ exception |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | exr -1 | display most recent exception | | .exr 7c901230 | display exception at address 7c901230 | | !cppexr 7c901230 | display c++ exception at address 7c901230 | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **10) Loaded modules and image information** | | |
| **Cmd** | **Variants / Params** | **Description** |
| lm | lm[ v | l | k | u | f ] *[m Pattern]*  lmD | List modules; verbose | with loaded symbols | k-kernel or u-user only symbol info | image path; pattern that the module name must match  DML mode of lm; lmv command links included in output |
| !dlls | !dlls  !dlls -i  !dlls -l  !dlls -m  !dlls -v  !dlls -c ModuleAddr  !dlls -? | all loaded modules with **load count**  by initialization order  by load order (default)  by memory order  with version info  only module at ModuleAddr  brief help |
| !imgreloc | *ImgBaseAddr* | information about relocated images |
| !lmi | *Module* | detailed info about a module (including exact symbol info) |
| !dh | !dh *ImgBaseAddr*  !dh -f *ImgBaseAddr*  !dh -s *ImgBaseAddr*  !dh -h | Dump headers for ImgBaseAddr  f = file headers only  s = section headers only  h = brief help   The !lmi extension extracts the most important information from the image header and displays it in a concise summary format. It is often more useful than !dh. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | lm | display all loaded and unloaded modules | | lmv m kernel32 | display verbose (all possible) information for kernel32.dll | | lmD | DML variant of lm | |  | | | !dlls -v -c kernel32 | display information for kernel32.dll, including **load-count** | | !lmi kernel32 | display detailed information about kernel32, including **symbol information** | | !dh kernel32 | display headers for kernel32 | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **11) Process related information** | | |
| **Cmd** | **Variants / Params** | **Description** |
| !dml\_proc |  | (DML) displays current processes and allows drilling into processes for more information |
| | (pipe) |  | Print status of all processes being debugged |
| .tlist |  | lists all processes running on the system |
| !peb |  | display formatted view of the process's environment block (PEB) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | !peb | Dump formatted view of processes PEB (only some information) | | r $peb | Dump address ob PEB. $peb == pseudo-register | | dt ntdll!\_PEB | Dump PEB struct | | dt ntdll!\_PEB @$peb -r | Recursively (-r) dump PEB of our process | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **12) Thread related information** | | |
| **Cmd** | **Variants / Params** | **Description** |
| ~ | ~  ~\* *[Command]*  ~. *[Command]*  ~# *[Command]*  ~Number *[Command]*  ~~[TID] *[Command]*  ~Ns | list threads  all threads  current thread  thread that caused the current event or exception  thread whose ordinal is Number  thread whose thread ID is TID (the brackets are required)  switch to thread N (new current thread)   *[Command]: works for a few regular commands such as k, r* |
| ~e | ~\* *e CommandString*  ~. *e CommandString*  ~# *e CommandString*  ~Number *e CommandString* | Execute thread-specific commands (CommandString = one or more commands to be executed) for:  all threads  current thread  thread which caused the current event  thread with ordinal |
| ~f | ~Thread f | Freeze thread (see ~ for Thread syntax) |
| ~u | ~Thread u | Unfreeze thread (see ~ for Thread syntax) |
| ~n | ~Thread n | Suspend thread = increment thread's suspend count |
| ~m | ~Thread m | Resume thread = decrement thread's suspend count |
| !teb |  | display formatted view of the thread's environment block (TEB) |
| !tls | !tls -1  !tls SlotIdx  !tls [-1 | SlotIdx] TebAddr | -1 = dump all slots for current thread  SlotIdx = dump only specified slot  TebAddr = specify thread; if omitted, the current thread is used |
| .ttime |  | display thread times (user + kernel mode) |
| !runaway | [Flags: 0 | 1 | 2] | display information about time consumed by each thread (0-user time, 1-kernel time, 2-time elapsed since thread creation). quick way to find out which threads are spinning out of control or consuming too much CPU time |
| !gle | !gle  !gle -all | Dump last error for current thread  Dump last error for all threads   Point of interest:  **SetLastError( dwErrCode )** checks the value of kernel32!g\_dwLastErrorToBreakOn and possibly executes a DbgBreakPoint.   **if** ((g\_dwLastErrorToBreakOn != 0 ) && (**dwErrCode == g\_dwLastErrorToBreakOn**))  **DbgBreakPoint();**   The downside is that SetLastError is only called from within KERNEL32.DLL.  Other calls to SetLastError are redirected to a function located in NTDLL.DLL, RtlSetLastWin32Error. |
| !error | !error ErrValue  !error ErrValue 1 | Decode and display information about an error value  Treat ErrValue value as an NTSTATUS code |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | ~\* k | call stack for all threads ~ !uniqstack | | ~2 f | Freeze Thread TID=2 | | ~# f | Freeze the thread causing the current exception | | ~3 u | Unfreeze Thread TID=3 | | ~2e r; k; kd | == ~2r; ~2k; ~2kd | | ~\*e !gle | will repeat every the extension command !gle for every single thread being debugged | |  | | | !tls -1 | Dump all TLS slots for current thread | |  | | | !runaway 7 | 1 (user time) + 2 (kernel time) + 4 (time elapsed since thread start) | |  | | | !teb | Dump formatted view of our threads TEB (only some information) | | dt ntdll!\_TEB @$teb | Dump TEB of current thread | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **13) Breakpoints** | | |
| **Cmd** | **Variants / Params** | **Description** |
| bl |  | List breakpoints |
| bc | bc \*  bc # [#] [#] | Clear all breakpoints  Clear breakpoint # |
| be | be \*  be # [#] [#] | Enable all bps  Enable bp # |
| bd | bd \*  bd # [#] [#] | Disable all bps  Disable bp # |
| bp | bp [Addr]  bp [Addr] ["CmdString"]   [~Thrd] bp[#] [Options] [Addr] [Passes] ["CmdString"] | Set breakpoint at address  CmdString = Cmd1; Cmd2; .. Executed every time the BP is hit.   ~Thrd == thread that the bp applies too.  # = Breakpoint ID  Passes = Activate breakpoint after #Passes (it is ignored before) |
| bu | bu [Addr]   See bp .. | Set unresolved breakpoint. bp is set when the module gets loaded |
| bm | bm **SymPattern**  bm SymPattern ["CmdString"]   [~Thrd] bm [Options] SymPattern [#Passes] ["CmdString"] | Set symbol breakpoint. SymPattern can contain wildcards  CmdString = Cmd1; Cmd2; .. Executed every time the BP is hit.   ~Thrd == thread that the bp applies too.  Passes = Activate breakpoint after #Passes (it is ignored before)   The syntax **bm SymPattern** is equivalent to using **x SymPattern** and then using bu on each of the results. |
| ba | ba [r|w|e] [Size] Addr   [~Thrd] ba[#] [r|w|e] [Size] [Options] [Addr] [Passes] ["CmdString"] | Break on Access: [r=read/write, w=write, e=execute], Size=[1|2|4 bytes]   [~Thrd] == thread that the bp applies too.  # = Breakpoint ID  Passes = Activate breakpoint after #Passes (it is ignored before) |
| br | br OldID NewID [OldID2 NewID2 ...] | renumbers one or more breakpoints |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse  With bp, the breakpoint location is always converted to an address. In contrast, a bu or a bm breakpoint is always associated with the symbolic value.    **Simple Examples**   |  |  | | --- | --- | | bp `mod!source.c:12` | set breakpoint at specified source code | | bm myprogram!mem\* | SymbolPattern is equivalent to using x SymbolPattern | | bu myModule!func | bp set as soon as myModule is loaded | | ba w4 77a456a8 | break on write access | | bp @@( MyClass::MyMethod ) | break on methods (useful if the same method is overloaded and thus present on several addresses) |   **Breakpoitns with options**   |  | | --- | | **Breakpoint that is triggered only once** | | bp mod!addr /1 | | **Breakpoint that will start hitting after k-1 passes** | | bp mod!addr k |   **Breakpoints with commands:** The command will be executed when the breakpoint is hit.   |  | | --- | | **Produce a log every time the breakpoint is hit** | | ba w4 81a578a8 "k;g" | | **Create a dump every time BP is hit** | | bu myModule!func ".dump c:\dump.dmp; g" | | **DllMain called for MYDLL -> check reason** | | bu MYDLL!DllMain "j (dwo(@esp+8) == 1) '.echo MYDLL!DllMain -> DLL\_PROCESS\_ATTACH; kn' ; 'g' " | | **LoadLibraryExW( anyDLL ) called -> display name of anyDLL** | | bu kernel32!LoadLibraryExW ".echo LoadLibraryExW for ->; du dwo(@esp+4); g" | | **LoadLibraryExW( MYDLL ) called? -> Break only if LoadLibrary is called for MyDLL** | | bu kernel32!LoadLibraryExW ";as /mu ${/v:MyAlias} poi(@esp+4); .if ( $spat( \"${MyAlias}\", \"\*MYDLL\*\" ) != 0 ) { kn; } .else { g }"   * The first parameter to LoadLibrary (at address ESP + 4) is a string pointer to the DLL name in question. * The MASM $spat operator will compare this pointer to a predefined string-wildcard, this is \*MYDLL\* in our example. * Unfortunately $spat can accept aliases or constants, but no memory pointers. This is why we store our string in question to an alias (MyAlias) first. * Our kernel32!LoadLibraryExW breakpoint will hit only if the pattern compared by $spat matches. Otherwise the application will continue executing. | | **Skip execution of a function** | | bu sioctl!DriverEntry "r eip = poi(@esp); r esp = @esp + 0xC; .echo sioctl!DriverEntry skipped; g"   * Right at a function’s entry point the value found on the top of the stack contains the return address  r eip = poi(@esp) -> Set EIP (instruction pointer) to the value found at offset 0x0 * DriverEntry has 2x4 byte parameters = 8 bytes + 4 bytes for the return address = 0xC  r esp = @esp + 0xC -> Add 0xC to Esp (the stack pointer), effectively unwinding the stack pointer | | bu MyApp!WinMain "r eip = poi(@esp); r esp = @esp + 0x14; .echo WinSpy!WinMain entered; g"   * WinMain has 4x4 byte parameters = 0x10 bytes + 4 bytes for the return address = 0x14 |   **Howto set a brekpoint in your code programatically?**   * kernel32!DebugBreak * ntdll!DbgBreakPoint * \_\_asm int 3 (x86 only) |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **14) Tracing and stepping (F10, F11)   Each step executes either a single assembly instruction or a single source line, depending on whether the debugger is in assembly mode or source mode.  Use the l+t and l-t commands or the buttons on the WinDbg toolbar to switch between these modes.** | | |
| **Cmd** | **Variants / Params** | **Description** |
| g (F5) | g  gu | Go (F5)  Go up = execute until the current function is complete  gu ~= g @$ra  gu ~= bp /1 /c @$csp @$ra;g  -> $csp = same as esp on x86  -> $ra = The return address currently on the stack |
| p (F10) | p   pr  p **Count**  p [Count] **"Command"**  p =StartAddress [Count] ["Command"]   [~Thread] p [=StartAddress] [Count] ["Command"] | **Single step** - executes a single instruction or source line. Subroutines are treated as a single step.   Toggle display of registers and flags  Count = count of instructions or source lines to step through before stopping  Command = debugger command to be executed after the step is performed  StartAddress = Causes execution to begin at the specified address. Default is the current EIP.   ~Thread = The specified thread is thawed and all others frozen |
| t (F11) | t  .. | **Single trace** - executes a single instruction or source line. For subroutines each step is traced as well. |
| pt | pt  .. | **Step to next return** - similar to the GU (go up), but staying in context of the current function  If EIP is already on a **return** instruction, the entire return is executed. After this return is returned, execution will continue until another **return** is reached. |
| tt | tt  .. | **Trace to next return** - similar to the GU (go up), but staying in context of the current function  If EIP is already on a **return** instruction, the debugger traces into the return and continues executing until another **return** is reached. |
| pc | pc  .. | **Step to next call** - executes the program until a call instruction is reached  If EIP is already on a **call** instruction, the entire call will be executed. After this call is returned execution will continue until another **call** is reached. |
| tc | tc  .. | **Trace to next call** - executes the program until a call instruction is reached  If EIP is already on a **call** instruction, the debugger will trace into the call and continue executing until another **call** is reached. |
| pa | pa StopAddr    par  pa StopAddr **"Command"**  pa **=StartAddress** StopAddr ["Command"] | **Step to address**; StopAddr = address at which execution will stop  Called functions are treated as a single unit   Toggle display of registers and flags  Command = debugger command to be executed after the step is performed  StartAddress = Causes execution to begin at the specified address. Default is the current EIP. |
| ta | ta StopAddr  .. | **Trace to address**; StopAddr = address at which execution will stop  Called functions are traced as well |
| wt | wt   wt [Options] [= StartAddr] [EndAddr]  wt -l Depth ..  wt -m Module [-m Module2] ..  wt -i Module [-i Module2] ..  wt -oa ..  wt -or ..  wt -oR ..  wt -nc ..  wt -ns ..  wt -nw .. | **Trace and watch data**. Go to the beginning of a function and do a **wt**. It will run through the entire function and display statistics.   StartAddr = execution begin; EndAddr = address at which to end tracing (default = after RET of current function)  l = maximum depth of traced calls  m = restrict tracing to Module  i = ignore code from Module  oa = dump actual address of call sites  or = dump return register values (EAX value) of sub-functions  oR = dump return register values (EAX value) in the appropriate type  nc = no info for individual calls  ns = no summary info  ns = no warnings |
| .step\_filter | .step\_filter  .step\_filter "FilerList"  .step\_filter /c | Dump current filter list = functions that are skipped when tracing (t, ta, tc)  FilterList = Filter 1; Filter 2; ... symbols associated with functions to be stepped over (skipped)  clear the filter list   .step\_filter is not very useful in assembly mode, as each function call is on a different line. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | g | go | | g `:123`; ? poi(counter); g | executes the current program to source line 123; print the value of counter; resume execution | |  | | | p | single step | | pr | toggle displaying of registers | | p 5 "kb" | 5x steps, execute "kb" thereafter | |  | | | pc | step to next CALL instruction | | pa 7c801b0b | step until 7c801b0b is reached | |  | | | wt | trace and watch sub-functions | | wt -l 4 -oR | trace sub-functions to depth 4, display their return values | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **15) Call stack** | | |
| **Cmd** | **Variants / Params** | **Description** |
| k | k [n] [f] [L] [#Frames]  kb ...  kp ...  kP ...  kv ... | dump stack; n = with frame #; f = distance between adjacent frames; L = omit source lines; number of stack frames to display  first 3 params  all params: param type + name + value  all params formatted (new line)  FPO info, calling convention |
| kd | kd [WordCnt] | display raw stack data + possible symbol info == dds esp |
| kM |  | DML variant with links to .frame #;dv |
| .kframes |  | Set stack length. The default is 20 (0x14). |
| .frame | .frame  .frame #  .frame /r [#] | show current frame  specify frame #  show register values   The .frame command specifies which local context (scope) will be used to interpret local variables, or displays the current local context.  When executing a near call, the processor pushes the value of the EIP register (which contains the offset of the instruction following the CALL instruction) onto the stack (for use later as a return-instruction pointer). This is the first step in building a frame. Each time a function call is made, another frame is created so that the called function can access arguments, create local variables, and provide a mechanism to return to calling function. The composition of the frame is dependant on the function calling convention. |
| !uniqstack | !uniqstack  !uniqstack [b|v|p] [n]  !uniqstack -? | show stacks for all threads  [b = first 3 params, v = FPO + calling convention, p = all params: param type + name + value], [n = with frame #]  brief help |
| !findstack | !findstack Symbol  !findstack Symbol [0|1|2]  !findstack -? | locate all stacks that contain Symbol or module  [0 = show only TID, 1 = TID + frames, 2 = entire thread stack]  brief help |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | k | display call stack | | kn | call stack with frame numbers | | kb | display call stack with first 3 params | | kb 5 | display first 5 frames only |   To get more than 3 Function Arguments from the stack  dd ChildEBP+8 (Parameters start at ChildEBP+8)  dd ChildEBP+8 (frame X) == dd ESP (frame X-1)   |  |  | | --- | --- | | !uniqstack | get all stacks of our process (one for each thread) | | !findstack kernel32 2 | display all stacks that contain "kernel32" | |  | | | .frame | show current frame | | .frame 2 | set frame 2 for the local context | | .frame /r 0d | display registers in frame 0 | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **16) Registers** | | |
| **Cmd** | **Variants / Params** | **Description** |
| r | r  r **Reg1, Reg2**  r Reg=**Value**   r Reg**:Type**             r Reg:[Num]Type    ~Thread r [Reg:[Num]Type] | Dump all registers  Dump only specified registers (i.e.: **r eax, edx**)  Value to assign to the register (i.e.: **r eax=5, edx=6**)   Type = data format in which to display the register (i.e.: **r eax:uw**)  ib = Signed byte  ub = Unsigned byte  iw = Signed word (2b)  uw = Unsigned word (2b)  id = Signed dword (4b)  ud = Unsigned dword (4b)  iq = Signed qword (8b)  uq = Unsigned qword (8b)  f = 32-bit floating-point  d = 64-bit floating-point   Num = number of elements to display (i.e.: **r eax:1uw**)  Default is full register length, thus **r eax:uw** would display two values as EAX is a 32-bit register.   Thread = thread from which the registers are to be read (i.e.: **~1 r eax**) |
| rM | rM Mask  rM Mask Reg1, Reg2  rM Mask Reg=Value  .. | Dump register types specified by Mask  Dump only specified registers from current mask  Value to assign to the register   Flags for Mask  0x1 = basic integer registers  0x4 = floating-point registers == rF  0x8 = segment registers  0x10 = MMX registers  0x20 = Debug registers  0x40 = SSE XMM registers == rX |
| rF | rF  rF Reg1, Reg2  rF Reg=Value  .. | Dump all floating-point registers == rM 0x4  Dump only specified floating-point registers  Value to assign to the register |
| rX | rX  rX Reg1, Reg2  rX Reg=Value  .. | Dump all SSE XMM registers == rM 0x40  Dump only specified SSE XMM registers  Value to assign to the register |
| rm | rm  rm ?  rm Mask | Dump default register mask. This mask controls how registers are displayed by the "r".  Dump a list of possible Mask bits  Specify the mask to use when displaying the registers. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | rm ? | show possible bit mask | | rm 1 | enable integer registers only | | r | dump all integer registers | | r eax, edx | dump only eax and edx | | r eax=5, edx=6 | assign new values to eax and edx | | r eax:1ub | dump only the first byte from eax | |  | | | rm 0x20 | enable debug register mask | | r | dump debug registers | |  | | | rF | dump all floating point register | | rM 0x4 | dump all floating point register | | rm 0x4; r | dump all floating point registers | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **17) Information about variables** | | |
| **Cmd** | **Variants / Params** | **Description** |
| dt | dt -h  dt [mod!]**Name**  dt [mod!]Name **Field** [Field]  dt [mod!]Name [Field] **Addr**  dt [mod!]Name**\***   dt [**-n|y**] [mod!]Name [**-n|y**] [Field] [Addr]    dt [-n|y] [mod!]Name [-n|y] [Field] [Addr] **-abcehioprsv** | Brief help  Dump variable info  Dump only 'field-name(s)' (struct or unions)  Addr of struct to be dumped  list symbols (wildcard)   -n Name = param is a name (use if name can be mistaken as an address)  -y Name = partially match instead of default exact match   **-a** = Shows array elements in new line with its index  -b = Dump only contiguous block of struct  -c = Compact output (all fields in one line)  -i = Does not indent the subtypes  -l ListField = Field which is pointer to the next element in list  -o = Omit the offset value (fields of struct)  -p = Dump from physical address  **-r[l]** = Recursively dump subtypes/fields (up to l levels)  -s [size] = For enumeration only, enumerate types only of given size.  **-v** = Verbose output. |
| dv | dv  dv **Pattern**  dv [**/i /t /V**] [Pattern]  dv [/i /t /V **/a /n /z**] [Pattern] | display local variables and parameters  vars matching Pattern  i = type (local, global, parameter), t = data type, V = memory address or register location  a = sort by Addr, n = sort by name, z = sort by size |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | dt ntdll!\_PEB\* | list all variables that contain the word \_PEB | | dt ntdll!\_PEB\* -v | list with verbose output (address and size included) | | dt ntdll!\_PEB\* -v -s 9 | list only symbols whose size is 9 bytes | |  | | | dt ntdll!\_PEB | dump \_PEB info | | dt ntdll!\_PEB @$peb | dump \_PEB for our process | | dt ntdll!\_PEB 7efde000 | dump \_PEB at Addr 7efde000  You can get our process's PEB address with "r @$peb" or with "!peb". | | dt ntdll!\_PEB Ldr SessionId | dump only PEB's Ldr and SessionId fields | | dt ntdll!\_PEB Ldr -y OS\* | dump Ldr field + all fields that start with OS\* | |  | | | dt mod!var m\_cs. | dump m\_cs and expand its subfields | | dt mod!var m\_cs.. | expand its subfields for 2 levels | |  | | | dt ntdll!\_PEB -r2 | dump recursively (2 levels) | |  | | | dv /t /i /V | dump local variables with type information (/t), addresses and EBP offsets (/V), classify them into categories (/i)  Note: dv will also display the value of a THIS pointer for methods called with the "this calling-convention".  BUG: You must first execute a few commands before dv displays the correct value.  Right at a function's entry point the THIS pointer is present in ECX, so you can easily get it from there. | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **18) Memory** | | |
| **Cmd** | **Variants / Params** | **Description** |
| d\* | d[a| u| b| w| W| d| c| q| f| D] [/c #] [Addr]               dy[b | d] .. | **Display memory** [#columns to display]  a = ascii chars  u = Unicode chars   b = byte + ascii  w = word (2b)  W = word (2b) + ascii  d = dword (4b)  c = dword (4b) + ascii  q = qword (8b)   f = floating point (single precision - 4b)  D = floating point (double precision - 8b)   b = binary + byte  d = binary + dword |
| e\* | e[ b | w | d | q | f | D ] Addr Value          e[ a | u | za | zu ] Addr "String" | **Edit memory**  b = byte  w = word (2b)  d = dword (4b)  q = qword (8b)   f = floating point (single precision - 4b)  D = floating point (double precision - 8b)   a = ascii string  za = ascii string (NULL-terminated)  u = Unicode string  zu = Unicode string (NULL-terminated) |
| ds, dS | ds *[/c #] [Addr]*  dS *[/c #] [Addr]* | **Dump string struct** (struct! not null-delimited char sequence)  s = STRING or ANSI\_STRING  S = UNICODE\_STRING |
| d\*s | dds [/c #] [Addr]  dqs [/c #] [Addr] | **Display words and symbols** (memory at Addr is assumed to be a series of addresses in the symbol table)  dds = dwords (4b)  dqs = qwords (8b) |
| dd\*, dq\*, dp\* | dd\*  dq\*  dp\*    d\*a  d\*u  d\*p | **Display referenced memory** = display pointer at specified Addr, dereference it, and then display the memory at the resulting location in a variety of formats.   the 2nd char determines the pointer size used:  dd\* -> 32-bit pointer used  dq\* -> 64-bit pointer used  dp\* -> standard size: 32-bit or 64-bit, depending on the CPU architecture   the 3rd char determines how the dereferenced memory is displayed:  d\*a -> dereferenced mem as asci chars  d\*u -> dereferenced mem as Unicode chars  d\*p -> dereferenced mem as dword or qword, depending on the CPU architecture. If this value matches any known symbol, this symbol is displayed as well. |
| dl | dl[b] Addr MaxCount Size | **Display linked list** (LIST\_ENTRY or SINGLE\_LIST\_ENTRY)  b = dump in reverse order (follow BLinks instead of FLinks)  Addr = start address of the list  MaxCount = max # elements to dump  Size = Size of each element   Use !list to execute some command for each element in the list. |
| !address | !address -?  !address Addr  !address -summary  !address -RegionUsageXXX | Display info about the memory used by the target process  Brief help  Dump info for region with Addr  Dump summary info for process  Dump specified regions (RegionUsageStack, RegionUsagePageHeap, ..) |
| !vprot | !vprot -?  !vprot Addr | Brief Help  Dump virtual memory protection info |
| !mapped\_file | !mapped\_file -?  !mapped\_file Addr | Brief Help  Dump name of the file containing given Addr |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | dd 0046c6b0 | display dwords at 0046c6b0 | | dd 0046c6b0 L1 | display 1 dword at 0046c6b0 | | dd 0046c6b0 L3 | display 3 dwords at 0046c6b0 | | du 0046c6b0 | display Unicode chars at 0046c6b0 | | du 0046c6b0 L5 | display 5 Unicode chars at 0046c6b0 | |  | | | dds esp == kd | display words and symbols on stack | |  | | | !mapped\_file 00400000 | Dump name of file containing address 00400000 | |  | | | !address | show all memory regions of our process | | !address -RegionUsageStack | show all stack regions of our process | | !address esp | show info for committed sub-region for our thread's stack.  Note: For stack overflows SubRegionSize (size of committed memory) will be large, i.e.:  AllocBase : SubRegionBase - **SubRegionSize**  ---------------------------------------------  001e0000 : 002d6000 - 0000a000 |   **Determine stack usage for a thread**    Stack Identifier Memory Identifier ^  ---------------------------------------------------------  -------------- <- \_TEB.StackBase SubRegionBase3 + SubRegionSize3  | |  | MEM\_COMMIT |  | |  |------------| <- \_TEB.StackLimit SubRegionBase3 ^, SubRegionBase2 + SubRegionSize2  | PAGE\_GUARD |  |------------| SubRegionBase2 ^, SubRegionBase1 + SubRegionSize1  | |  |MEM\_RESERVED|  | |  |------------| <- \_TEB.DeallocationStack AllocationBase or RegionBase, SubRegionBase1 ^    DeallocationStack: dt ntdll!\_TEB TebAddr DeallocationStack    From MSDN CreateThread > dwStackSize > "Thread Stack Size":  "Each new thread receives its own stack space, consisting of both committed and reserved memory. By default, each thread uses 1 Mb of reserved memory, and one page of committed memory. The system will commit one page block from the reserved stack memory as needed." |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **19) Manipulating memory ranges** | | |
| **Cmd** | **Variants / Params** | **Description** |
| c | c Range DestAddr | Compare memory |
| m | m Range DestAddr | Move memory |
| f | f Range Pattern | Fill memory. Pattern = a series of bytes (numeric or ASCII chars) |
| s | s *Range Pattern*   s -[Flags]b *Range Pattern*    s -[Flags]w *Range 'Pattern'*  s -[Flags]d *Range 'Pattern'*  s -[Flags]q *Range 'Pattern'*    s -[Flags]a *Range "Pattern"*  s -[Flags]u *Range "Pattern"*    s -[Flags,l length]sa *Range*  s -[Flags,l length]su *Range*    s -[Flags]v *Range Object* | Search memory   b = byte (default value)  Pattern = a series of bytes (numeric or ASCII chars)   w = word (2b)  d = dword (4b)  q = qword (8b)  Pattern = enclosed in single quotation marks (for example, 'Tag7')   a = ascii string (must not be null-terminated)  u = Unicode string (must not be null-terminated)  Pattern = enclosed in double quotation marks (for example, "This string")   Search for any memory containing printable ascii strings  Search for any memory containing printable Unicode strings  Length = minimum length of such strings; the default is 3 chars   Search for objects of the same type.  Object = Addr of a pointer to the Object or of the Object itself   Flags  -------  w = search only writable memory  1 = output only addresses of search matches (useful if you are using the .foreach)  Flags must be surrounded by a single set of brackets without spaces.  Example: **s -[swl 10]Type Range Pattern** |
| .holdmem | .holdmem -a Range  .holdmem -o  .holdmem -c Range  .holdmem -D  .holdmem -d { Range | Address } | **Hold and compare memory.** The comparison is made byte-for-byte  Memory range to safe  Display all saved memory ranges  Compares Range to all saved memory ranges  Delete all saved memory ranges  Delete specified memory ranges (any saved range containing Addr or overlapping with Range) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | c Addr (Addr+100) DestAddr | compare 100 bytes at Addr with DestAddr | | c Addr L100 DestAddr | -||- | |  | | | m Addr L20 DestAddr | move 20 bytes from Addr to DestAddr | |  | | | f Addr L20 'A' 'B' 'C' | fill specified memory location with the pattern "ABC", repeated several times | | f Addr L20 41 42 43 | -||- | |  | | | s 0012ff40 L20 'H' 'e' 'l' 'l' 'o' | search memory locations 0012FF40 through 0012FF5F for the pattern "Hello" | | s 0012ff40 L20 48 65 6c 6c 6f | -||- | | s -a 0012ff40 L20 "Hello" | -||- | | s -[w]a 0012ff40 L20 "Hello" | search only writable memory | |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **20) Memory: Heap** | | |
| **Cmd** | **Variants / Params** | **Description** |
| !heap | !heap **-?**   !heap  !heap **-h**  !heap -h [HeapAddr | Idx | 0]  !heap -v [HeapAddr | Idx | 0]  !heap **-s** [HeapAddr | 0]  !heap -i [HeapAddr]  !heap -x [-v] Address  !heap -l | Brief help   List heaps with index and HeapAddr  List **heaps with index and range** (= startAddr(=HeapAddr), endAddr)  Detailed heap info [Idx = heap Idx, 0 = all heaps]  Validate heap [Idx = heap Idx, 0 = all heaps]  **Summary info, i.e. reserved and committed memory** [Idx = heap Idx, 0 = all heaps]  Detailed info for a block at given address  Search heap block containing the address (v = search the whole process virtual space)  Search for potentially leaked heap blocks |
| !heap -b, -B | !heap Heap -b [alloc | realloc | free] [Tag]  !heap Heap -B [alloc | realloc | free] | Set conditional breakpoint in the heap manager [Heap = HeapAddr | Idx | 0]  Remove a conditional breakpoint |
| !heap -flt | !heap -flt s Size  !heap -flt r SizeMin SizeMax | Dump info for allocations matching the specified size  Filter by range |
| !heap -stat | !heap **-stat**  !heap **-stat -h** [HeapHandle | 0] | Dump heap **handle list**  Dump usage statistic for every AllocSize [HeapHandle = given heap | 0 = all heaps].  The statistic includes AllocSize, #blocks, TotalMem for each AllocSize. |
| !heap -p | !heap -p -?  !heap **-p**  !heap -p -h HeapHandle  !heap **-p -a UserAddr**  !heap **-p -all** | Extended page heap help  Summary for NtGlobalFlag, HeapHandle + NormalHeap list \*\*  Detailed info about a page heap with Handle  Details of heap allocation containing UserAddr. Prints backtraces when available.  Details of all allocations in all heaps in the process.  The output includes UserAddr and AllocSize for every HeapAlloc call. |
| It seems that the following applies for windows XP SP2:   **a) Normal heap**   1. CreateHeap -> creates a \_HEAP 2. AllocHeap -> creates a \_HEAP\_ENTRY   **b) Page heap enabled** (gflags.exe /i MyApp.exe +hpa)   1. CreateHeap -> creates a \_DPH\_HEAP\_ROOT (+ \_HEAP + 2x \_HEAP\_ENTRY)\*\* 2. AllocHeap -> creates a \_DPH\_HEAP\_BLOCK   \*\* With page heap enabled there will still be a \_HEAP with two constant \_HEAP\_ENTRY's for every CreateHeap call.    |  |  |  | | --- | --- | --- | | **Term** | **Description** | **Heap type** | | **HeapHandle** | = value returned by **HeapCreate** or **GetProcessHeap**  For normal heap: HeapHandle == HeapStartAddr | Normal & page | | **HeapAddr** | = startAddr = NormalHeap | Normal & page | | **UserAddr, UserPtr** | = value in the range [**HeapAlloc**...HeapAlloc+AllocSize]  For normal heap this range is further within Heap[startAddr-endAddr] | Normal & page | | **UserSize** | = AllocSize (value passed to HeapAlloc) | Normal & page | |  | | | | \_HEAP | = HeapHandle = HeapStartAddr  For every **HeapCreate** a \_HEAP struct is created.  You can use "!heap -p -all" to get these addresses. | Normal heap | | \_HEAP\_ENTRY | For every **HeapAlloc** a \_HEAP\_ENTRY is created.  You can use "!heap -p -all" to get these addresses. | Normal heap | |  | | | | \_DPH\_HEAP\_ROOT | = usually HeapHandle + 0x1000  For every **HeapCreate** a \_DPH\_HEAP\_ROOT is created.  You can use "!heap -p -all" to get these addresses. | Page heap | | \_DPH\_HEAP\_BLOCK | For every **HeapAlloc** a \_DPH\_HEAP\_BLOCK is created.  You can use "!heap -p -all" to get these addresses. | Page heap | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse   |  |  | | --- | --- | | dt ntdll!\_HEAP | dump \_HEAP struct | | dt ntdll!\_DPH\_HEAP\_ROOT | dump \_DPH\_HEAP\_ROOT struct.  Enable page heap. Then you can use "!heap -p -all" to get addresses of actual \_DPH\_HEAP\_ROOT structs in your process. | | dt ntdll!\_DPH\_HEAP\_BLOCK | dump \_DPH\_HEAP\_BLOCK struct.  Enable page heap. Then you can use "!heap -p -all" to get addresses of actual \_DPH\_HEAP\_BLOCK structs in your process. | |  | | | !heap | list all heaps with index and HeapAddr | | !heap -h | list all heaps with range information (startAddr, endAddr) | | !heap -h 1 | detailed heap info for heap with index 1 | | !heap -s 0 | Summary for all heaps (reserved and committed memory, ..) | |  | | | !heap -flt s 20 | Dump heap allocations of size 20 bytes | |  | | | !heap -stat | Dump HeapHandle list. HeapHandle = value returned by HeapCreate or GetProcessHeap | | !heap -stat -h 00150000 | Dump usage statistic for HeapHandle = 00150000 | |  | | | !heap 2 -b alloc mtag | Breakpoint on HeapAlloc calls with TAG=mtag in heap with index 2 | |  | | | !heap -p | Dump heap handle list | | !heap -p -a 014c6fb0 | Details of heap allocation containing address 014c6fb0 + call-stack if available | | !heap -p -all | Dump details of all allocations in all heaps in the process |   **Who allocated memory - who called HeapAlloc?**   1. Select "Create user mode stack trace database" for your image in GFlags (gflags.exe /i MyApp.exe +ust) 2. From WinDbg's command line do a **!heap -p -a [UserAddr]**, where [UserAddr] is the address of your allocation \*\*\*. 3. While !heap -p -a [UserAddr] will dump a call-stack, no source information will be included. 4. To get source information you must additionally enable page heap in step 1 (gflags.exe /i MyApp.exe +ust +hpa) 5. Do a **dt ntdll!\_DPH\_HEAP\_BLOCK StackTrace [MyHeapBlockAddr]**, where [MyHeapBlockAddr] is the DPH\_HEAP\_BLOCK address retrieved in step 3. 6. Do a **dds [StackTrace]**", where [StackTrace] is the value retrieved in step 5.  Note that dds will dump the stack with source information included.   **Who created a heap - who called HeapCreate?**   1. Select "Create user mode stack trace database" and "Enable page heap" for your image in GFlags (gflags.exe /i MyApp.exe +ust +hpa) 2. a) From WinDbg's command line do a **!heap -p -h [HeapHandle]**, where [HeapHandle] is the value returned by **HeapCreate**. You can do a **!heap -stat** or **!heap -p** to get all heap handles of your process.  b) Alternatively you can use **!heap -p -all** to get addresses of all \_DPH\_HEAP\_ROOT's of your process directly. 3. Do a **dt ntdll!\_DPH\_HEAP\_ROOT CreateStackTrace [MyHeapRootAddr]**, where is the address of a \_DPH\_HEAP\_ROOT retrieved in step 2 4. Do a **dds**, where [CreateStackTrace] is the value retrieved in step 3.   **Finding memory leaks**   * From WinDbg's command line do a **!address –summary**.  If **RegionUsageHeap** or **RegionUsagePageHeap** are growing, then you might have a memory leak on the heap. Proceed with the following steps.  1. Enable "Create user mode stack trace database" for your image in GFlags (gflags.exe /i MyApp.exe +ust) 2. From WinDbg's command line do a **!heap -stat**, to get all active heap blocks and their handles. 3. Do a **!heap -stat -h 0**. This will list down handle specific allocation statistics for every AllocSize.  For every AllocSize the following is listed: AllocSize, #blocks, and TotalMem. Take the AllocSize with maximum TotalMem. 4. Do a **!heap -flt s [Size]**. [Size]=AllocSize that we determined in the previous step. This command will list down all blocks with that particular size. 5. Do a **!heap -p -a [UserAddr]** to get the stack trace from where you have allocated that much bytes. Use the [UserAddr] that you got in step 4. 6. To get source information you must additionally enable page heap in step 1 (gflags.exe /i MyApp.exe +ust +hpa) 7. Do a **dt ntdll!\_DPH\_HEAP\_BLOCK StackTrace [MyHeapBlockAddr]**, where [MyHeapBlockAddr] is the DPH\_HEAP\_BLOCK address retrieved in step 5. 8. Do a **dds [StackTrace]**", where [StackTrace] is the value retrieved in step 7.  Note that dds will dump the stack with source information included.   **\*\*\* What is a [UserAddr]?**   1. [UserAddr] is usually the address returned by HeapAlloc: 2. int AllocSyze = 0x100000; // == 1 MB   BYTE\* **pUserAddr** = (BYTE\*) HeapAlloc( GetProcessHeap(), HEAP\_ZERO\_MEMORY, **AllocSyze**);   1. Often any address in the range [UserAddr....UserAddr+AlloSize] is also a valid parameter:   !heap -p -a [**UserAddr....UserAddr+AlloSize**] |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **21) Application Verifier  Application Verifier profiles and tracks Microsoft Win32 APIs (heap, handles, locks, threads, DLL load/unload, and more), Exceptions, Kernel objects, Registry, File system. With the !avrf extension we get access to this tracking information!** | | |
| **Cmd** | **Variants / Params** | **Description** |
| !avrf |  | Displays Application Verifier options. If an Application Verifier Stop has occurred, reveal the nature of the stop and what caused it. |
| !avrf | -?   -vs *N*  -vs *-a ADDR*   -hp *N*  -hp *-a ADDR*   -cs *N*  -cs *-a ADDR*   -dlls *N*  -ex *N*  -cnt  -threads  -trm  -trace *INDEX*  -brk *[INDEX]* | Brief help   Dump last N entries from vspace log (MapViewOfFile, UnmapViewOfFile, ..).  Searches ADDR in the vspace log.   HeapAlloc, HeapFree, new, and delete log  Searches ADDR in the heap log.   DeleteCriticalSection API log (last #Entries). ~CCriticalSection calls this implicitly.  Searches ADDR in the critical section delete log.   LoadLibrary/FreeLibrary log  exception log  global counters (WaitForSingleObject, HeapAllocation calls, ...)  thread information + start parameters for child threads  TerminateThread API log  dump stack trace with INDEX.  dump or set/reset break triggers. |

[[Go up](http://windbg.info/doc/1-common-cmds.html#top) Go up](http://windbg.info/doc/1-common-cmds.html" \l "top)

|  |  |  |
| --- | --- | --- |
| **22) Logging extension (logexts.dll)   You must enable the following options for you image in GFlags:  -> "Create user mode stack trace database"  -> "Stack Backtrace: (Megs)" -> 10  -> It seems that you sometimes also need to check and specify the "Debugger" field in GFlags** | | |
| **Cmd** | **Variants / Params** | **Description** |
| !logexts.help |  | displays all Logexts.dll extension commands |
| !loge | !loge *[dir]* | Enable logging + possibly initialize it if not yet done. Output directory optional. |
| !logi |  | Initialize (=inject Logger into the target application) but don't enable logging. |
| !logd |  | Disable logging |
| !logo | !logo  !logo [e|d] [d|t|v] | List output settings  Enable/disable [d - Debugger, t - Text file, v - Verbose log] output. Use logviewer.exe to examine Verbose logs. |
| !logc | !logc  !logc p #  !logc [e|d] \*  !logc [e|d] # [#] [#] | List all categories  List APIs in category #  Enable/disable all categories  Enable/disable category # |
| !logb | !logb p  !logb f | Print buffer contents to debugger  Flush buffer to log files |
| !logm | !logm  !logm [i|x] [DLL] [DLL] | Display module inclusion/exclusion list  Specify module inclusion/exclusion list |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://windbg.info/templates/wiki-like-rk/cmd_specific/images/minus.gif Collapse  Enable 19-ProcessesAndThreads and 22-StringManipulation logging:   |  |  | | --- | --- | | !loge | Enable logging | | !logc d \* | Disable all categories | | !logc p 19 | Display APIs of category 19 | | logc e 19 22 | Enable category 19 and 22 | | !logo d v | Disable verbose output | | !logo d t | Disable text output | | !logo e d | Enable debugger output | |