WinDbg Cheat Sheet (user mode only)

Help Commands

`?`	Help on Debugee commands
`.help`	Help on Debugger commands
`.hh command`	Open WinDbg's help for this command

Execution Control

`restart`	Stop and restart execution
`t (F11)`	Step into (trace)
`p [count] (F10)`	Step over
`pa address`	Run to address
`pt`	Execute until a return instruction is reached
`ph [count]`	Execute until a branching instruction is reached. **count** = # of branches reached until it stops
`g (F5)`	Continue (go)
`gu (Shift-F11)`	Execute until the current function is complete
`(Ctrl-Break)`	Break

Breakpoints

`bl`	List breakpoints
`bp [addr] ["script"]`	Set a breakpoint
`bp`	Set breakpoint at current instruction
`bp addr`	Set breakpoint at specified address
`bp addr "script"`	Set a breakpoint and run script when hit `bp 403250 ".echo BP hit;g"`
`bu symbol`	Set unresolved breakpoint on a symbol
`bm pattern`	Set breakpoint on all symbols (unresolved by default) matching the specified pattern
`bm /d pattern`	Converts the breakpoints to addresses
`bc #`	Clear a breakpoint
`bc *`	Clear all breakpoints
`bd #`	Disable a breakpoint
`bd *`	Disable all breakpoints
`be #`	Enable a breakpoint
`be *`	Enable all breakpoints
`ba [access] [size] addr`	Set a breakpoint on memory access Size can be **1**, **2**, or **4** Access: **r** = Break on read acces **w** = Break on write access **e** = Break on execute access

Listing Modules

`lm [olfv]`	List all modules
`lm o`	List only loaded modules
`lm l`	List modules with symbol information
`lm f`	List all modules and their full image path
`lm v`	List all modules and be verbose
`lm a address`	Display the module that contains *address*
`lm m pattern`	Find module name, can contain wildcard
`lm M pattern`	Find image path, can contain wildcard

Symbols

`.reload /f`	Reload all symbols
`ld module`	Load symbols for a module
`ld *`	Load symbols for all modules
`ln address`	Find nearest symbol to address
`x module!symbol`	Display the symbols that match the specified pattern, can contain wildcard

Registers

`r`	Display all registers and their values
`r reg`	Display a single register and it's value
`r reg=value`	Set the register to a specific value

Unassembly

`u[ub] address [L#]`	Unassemble from memory
`uu address`	Disassembly continues past read error
`ub address`	Determine range by counting backwards
`u address L#`	Set the number of instructions to disassemble

Display Memory

`d{type} [/c#] addr [L#]`	Display the contents of memory. Types: **b** = Bytes + ASCII characters **w** = WORD (2 bytes) **W** = WORD + ASCII characters **d** = DWORD (4 bytes) **c** = DWORD + ASCII characters **q** = QWORD (8 bytes) **a** = ASCII string up until first null byte **u** = Unicode string up until first null byte **f** = Single-precision float numbers (4 bytes) **D** = Double-precision float numbers (8 bytes)
`d* /c# addr`	Set the number of columns to use in the display
`d* addr L#`	Set the length of output

Searching Memory

`s -[type] range pattern`	Search memory range for pattern. Types: **b** = Byte **w** = WORD (2 bytes) **d** = DWORD (4 bytes) **q** = QWORD (8 bytes) **a** = ASCII string **u** = Unicode string
`s -a 0 L?80000000 "string"`	Search entire user process memory space for a string. Must use "**L?**" if range is > 256 MB.
`s -d 0 L?80000000 41414141`	Search entire user process memory space for a DWORD value.

Display Type Information

`dt [-r] name`	Display variable or data type information
`dt -r name`	Recursively dump the subtype fields
`dt name addr`	Specify the address of the struct
`dt ntdll!_TEB @$teb`	Use @ to specify a register
`dt name field`	Specify the field to display

Display Memory & Symbols

`dds range`	Display DWORD (4 byte) values & symbols
`dqs range`	Display QWORD (8 byte) values & symbols
`dps range`	Display pointer-sized (4 or 8 byte) values & symbols

Evaluate Expressions

`? expr`	Evaluates an expression. Examples:
	`? 77269bc0 - 77231430`
	`? 77269bc0 >> 18`
	`? 41` (to see value in decimal)

`?? expr`	Evaluates C++ expression. Example: `?? sizeof(ntdll!_TEB)`
`.formats expr`	Evaluate and show in multiple formats

Prefixes

`0x`	Hexadecimal (default)
`0n`	Decimal
`0y`	Binary

Miscellaneous

`k[bpPv]`	Display stack backtrace **p** = Display all parameters passed to each function **P** = Same as **p** but printed on a second line
`a [address]`	Assemble x86 instructions and puts the resulting op codes into memory. ***address*** specifies the start of memory where the resulting codes are put.

Extensions

`!address [address]`	Display a memory map. ***address*** specifies the address for the region to display.
`!exchain`	Display the current exception handler chain
`!vprot address`	Display virtual memory protection information
`!dh address`	Display the headers for the specified image