

## **Microsoft WinDbg Debugger**

## Microsoft WinDbg Debugger





## **WinDbg Debugging Procedures**

[Running WinDbg from the command line](#)

[Building a Program for WinDbg](#)

[Using Workspaces](#)

[Setting Breakpoints](#)

[Controlling Program Execution](#)

[Using Debugging Information Windows](#)

[Using Watch Expressions](#)

[Using Quickwatch](#)

[Moving to a line in a file](#)

[Moving to a specific Address or Function](#)

[Viewing the Call Stack](#)

[Choosing Processes](#)

[Choosing Threads](#)

[Using the Command Window](#)

[Debugging Remotely](#)

[Postmortem Debugging](#)

[Setting Debugging Options](#)





## Command Window Reference

This section describes the commands that you can run in the Command window. It describes the commands that apply to debugging user-mode applications, as well as those that apply to debugging kernel-mode drivers.

You can use standard editing keys when you enter a command. Use the Up- and Down-Arrow keys to retrieve previous commands. Edit the current command line with the Backspace, Delete, Insert, and Left- and Right-Arrow keys. Press the Esc key to clear the current line.

To copy text from a previous line in the Command window, highlight the text by holding down the left mouse button and dragging the mouse pointer. Click the right mouse button to copy the highlighted text to the Clipboard. Click the right button again to paste the text into the current command line. You can use the same technique to copy text from other windows to the Command window. You can also press CTRL+INS to copy highlighted text to the Clipboard and press SHIFT+INS to paste text from the Clipboard to the insertion point.

You must use the correct [Range Syntax](#) or [Process and Thread Syntax](#) with many of the commands. For some commands, you must specify a [context](#).

Command	Definition
*	<a href="#">Comment</a>
;	<a href="#">Command Separator</a>
?	<a href="#">Evaluate Expression</a>
!	<a href="#">User Extension DLL</a>
	<a href="#">Display Process State</a>
~	<a href="#">Display Thread State</a>
#	<a href="#">Search for Disassembly Pattern</a>
%	<a href="#">Change Context</a>
.ATTACH	<a href="#">Attach to Process</a>
.CACHE	<a href="#">Cache Size</a>
.LIST	<a href="#">Display Source/Assembly Listing</a>
.LOGAPPEND	<a href="#">Append Log File</a>
.LOGCLOSE	<a href="#">Close Log File</a>
.LOGOPEN	<a href="#">Open Log File</a>
.REBOOT	<a href="#">Reboot Target Machine</a>
.RELOAD	<a href="#">Reload Symbols</a>
BC	<a href="#">Breakpoint Clear</a>
BD	<a href="#">Breakpoint Disable</a>
BE	<a href="#">Breakpoint Enable</a>
BL	<a href="#">Breakpoint List</a>
BP	<a href="#">Set Breakpoint</a>
C	<a href="#">Compare Memory</a>
DA, DB, DC, DD, DI, DS, DT, DU, DW	<a href="#">Display Memory</a>
EA, EB, ED, EI, ES, ET, EU, EW	<a href="#">Enter Values</a>

F	<a href="#"><u>Freeze Thread</u></a>
FIA, FIB, FID, FII, FIS, FIT, FIU, FIW	<a href="#"><u>Fill Memory</u></a>
FR	<a href="#"><u>Floating-Point Registers</u></a>
G	<a href="#"><u>Go</u></a>
GH	<a href="#"><u>Go-- Exception Handled</u></a>
GN	<a href="#"><u>Go-- Exception not Handled</u></a>
K, KB, KN, KS, KV	<a href="#"><u>Display Stack Backtrace</u></a>
L	<a href="#"><u>Restart Debuggee</u></a>
LM	<a href="#"><u>List Loaded Modules</u></a>
LN	<a href="#"><u>List Nearest Symbols</u></a>
M	<a href="#"><u>Move Memory</u></a>
N	<a href="#"><u>Set Number Base</u></a>
P	<a href="#"><u>Program Step</u></a>
Q	<a href="#"><u>Quit WinDbg</u></a>
R	<a href="#"><u>Registers</u></a>
REMOTE	<a href="#"><u>Start Remote Server</u></a>
RT	<a href="#"><u>Register Display Toggle</u></a>
SA, SB, SD, SI, SS, ST, SU, SW	<a href="#"><u>Search Memory</u></a>
S+, S-	<a href="#"><u>Set Source/Assembly Mode</u></a>
SEB, SEW	<a href="#"><u>Set Error Break, Set Error Warning</u></a>
SX, SXD, SXE, SXN	<a href="#"><u>Set Exceptions</u></a>
T	<a href="#"><u>Trace</u></a>
U	<a href="#"><u>Unassemble</u></a>
X	<a href="#"><u>Examine Symbols</u></a>
Z	<a href="#"><u>Unfreeze Thread</u></a>



### Range Syntax

You can specify a range as either of the following:

*startaddress endaddress*

*startaddress L length*

With the first method, you specify the start and end addresses of the range. With the second method, you specify the start address, the letter 'L', and the number of data items in the range.

### Note

With the DC (Display Code) and the U (Unassemble) commands, you can also use the following to dump *line* instructions starting at *startaddress*:

*startaddress L line*



## Process and Thread Syntax

You can specify a process in the following ways:

Symbol	Description
.	The current process.
<i>number</i>	The process number.
*	All processes.

You can specify a thread in the following ways:

Symbol	Description
~.	The current thread.
~ <i>number</i>	The thread ID.
~*	All threads.

If you do not specify a process or thread with a command, the command will act on the current process or thread. To set the current process or thread, enter a process or thread specifier alone on a command line.

To display the state of the processes or threads, type pipeline (!) or tilde (~) alone on the command line.

In kernel debugging, threads represent the physical processors of the target machine: a two-processor machine would have two threads, one for each processor. The single process represents the target machine itself.





**\* (Comment)**

### **Syntax**

\*

### **Description**

If this character is at the start of a line then the rest of the line is treated as a comment.



**; (Command Separator)**

### Syntax

;

### Description

Use this character to separate multiple commands on a single line. Commands are executed sequentially from left to right. All commands on a single line refer to the current thread, unless otherwise specified. If a command causes the thread to execute, the remaining commands on the line will be deferred until that thread stops on a debug event.

### Examples

```
P;?szVar;P;?szVar
```

Step twice, printing the variable "szVar" after each step.

```
g@123;?counter;g
```

Go until source line 123, print the value of "counter", then go.



## ? (Evaluate Expression)

### Description

Evaluates and displays the value of the expression or symbol in the context of the current thread and process.

### Note

The C++ expression evaluator cannot calculate expressions that use the conditional operator (?:).

### Syntax

? *expression* [, *format*]

### Parameters

*expression*

Expression to be evaluated with the expression evaluator.

*format*

C-style format characters. For example, to display an int, use "i" as the format specifier. Do not use percent signs (%).

Formats include:

Format	Definition	Example
D, d, I, i	Signed decimal integer.	> ?0x100,i 256
u	Unsigned decimal integer.	> ?-1,u 4294967295
o	Octal integer.	> ?40000,o 0o00001000000
x, X	Hexadecimal integer.	> ?40000,x 0x00040000
f	Signed value in floating point format with six decimal places (takes implicit/explicit type FLOAT as an argument).	> ?(float)(3./2.),f 1.500000
e, E	Signed value in scientific notation with up to six decimal places (trailing zeros and decimal point are truncated).	> ?(float)(3./2.),e 1.500000e+000
g, G	Signed value in floating point decimal format or scientific format, whichever is more compact.	> ?(float)(3./2.),g 1.5
c	Single character.	> ?41,c 'A'
s	Characters up to the first null character.	> ?szHello,s Hello (szHello="Hello")
l	Long-Word modifier.	> ?3./2.,lf 1.500000
L	Long-Long Word modifier	> ?0xffffffff,Li -1

h	Half-Word modifier:	> ?0x12345,hi 9029
---	---------------------	-----------------------

**Note:**

The "l", "L", and "h" modifiers work differently with different base types.

If, for instance, you are coercing an expression to type LONG DOUBLE (80 bit), the "L" modifier will work correctly on that expression. The "L" modifier also works correctly on LONG-LONG integer expressions, as shown in the table above.

WinDbg does not do implicit casting when evaluating an expression. It operates on the actual bit quantity unless coerced through casting, as shown above.



## ! (User Extension DLL)

### Syntax

! [[*filename.*]]*function* [[*string*]]

### Parameters

#### *filename*

Name of the DLL to call (without the .DLL extension), followed by a period (.). The default filename is NTSDEXTS.DLL.

In kernel debugging, the default extension DLL depends upon the target platform:

Target Platform	Default Extension DLL
x86	KDEXTX86.DLL
MIPS	KDEXTMIP.DLL
ALPHA	KDEXTALP.DLL

#### *function*

Name of the DLL function to call.

#### *string*

String to pass to the DLL function.

### Description

Calls a DLL function from WinDbg. Use this command to call special routines while debugging. For a complete list of the built-in extension commands, enter !? at the WinDbg prompt. See also [Kernel Debugging Extensions](#) and [Creating Extensions](#).



## # (Search for Disassembly Pattern)

### Syntax

# [[*pattern*]] [[*address*]]

### Parameters

*pattern*

The pattern to search for in the Disassembly window.

*address*

The address after which to search.

### Description

Displays the first line of assembly code after *address* that contains the specified pattern.



## % (Change Context)

### Syntax

%[[*frame*]]

### Parameters

*frame*

The number of the frame on the call stack to change context to. The current procedure is number 0 on the stack.

### Description

Changes the context to the specified frame. Updates the Locals window with the variables of the new frame.

Use the KN command to list the call stack with the frame numbers.



## **.ATTACH (Attach to Process)**

### **Syntax**

`.ATTACH process`

### **Parameters**

*process*

Process ID of task to debug.

### **Description**

Starts debugging the process specified by *process*. You can use PView to find process IDs of running programs or use the [Attach command](#) from the Run menu to select a task.

This command is similar to the /P command-line option.





## **.CACHE (Cache Size)**

### **Syntax**

`.CACHE [[cache size]]`

### **Parameters**

*cache size*

The size of the kernel debugging cache, in MB. If you do not specify *cache size*, the command displays the status of the cache. If you issue `.cache 0`, the cache is disabled.

### **Description**

Sets the size of the cache for WinDbg KD to use for memory values. Normal operations, such as single-stepping or the `g` command, invalidate the cache. If WinDbg KD has frozen the target machine, but hardware on the target can change memory (for example, through shared memory or DMA), you must disable the cache to see the memory changes. See also [Kernel Debugger Options](#).



## **.LIST (Display Source/Assembly Listing)**

### **Syntax**

`.LIST [[address]]`

### **Parameters**

*address*

The address at which to start displaying the listing.

### **Description**

Displays source-code lines and their corresponding assembly code, starting at *address*. If you do not specify *address*, the display starts from the current address.



## **.LOGAPPEND (Append Log File)**

### **Syntax**

`.LOGAPPEND [[filename]]`

### **Parameters**

*filename*

Filename of the log file. The default filename is WINDBG.LOG.

### **Description**

Sends a copy of the events and commands from the Command window to the specified log file. If you are already logging, the current log file will be closed. If *filename* already exists, new information will be appended to it.



## **.LOGCLOSE (Close Log File)**

### **Syntax**

.LOGCLOSE

### **Description**

Closes any currently open log file.



## **.LOGOPEN (Open Log File)**

### **Syntax**

`.LOGOPEN [[filename]]`

### **Parameters**

*filename*

Filename of the log file. The default filename is WINDBG.LOG.

### **Description**

Sends a copy of the events and commands from the Command window to the specified log file. If you are already logging, the current log file will be closed. If *filename* already exists, its contents will be overwritten.



## **.REBOOT (Reboot Target Machine)**

### **Syntax**

.REBOOT

### **Description**

Reboots the target machine (WinDbg KD only).



## **.RELOAD (Reload Symbols)**

### **Syntax**

`.RELOAD [[modulename]]`

### **Parameters**

*modulename*

The name of the module whose symbols you want to reload.

### **Description**

Loads symbols for the specified module on the target system. If you do not specify *modulename*, the symbols for all loaded modules will be reloaded. This command is useful in the event of a system crash, which can result in the loss of symbols for the target machine.



## **BC (Breakpoint Clear)**

### **Syntax**

BC {*breakpoint* [[*breakpoint...*]] | \*}

### **Parameters**

*breakpoint*

The number of the breakpoint to be cleared. Use the [Breakpoint List \(BL\)](#) command to display currently set breakpoints and their numbers. Specify ranges of breakpoints with a hyphen. Separate multiple breakpoints with spaces or commas.

### **Description**

Removes one or more previously set breakpoints from the system. BC\* clears all breakpoints.





## **BD (Breakpoint Disable)**

### **Syntax**

BD {*breakpoint* [[*breakpoint*...]] | \*} }

### **Parameters**

*breakpoint*

The number of the breakpoint to be disabled. Use the Breakpoint List (BL) command to display currently set breakpoints and their numbers. Specify ranges of breakpoints with a hyphen. Separate multiple breakpoints with spaces or commas.

### **Description**

Disables, but does not delete, one or more breakpoints. BD\* disables all breakpoints. While a breakpoint is disabled, the system does not check to see if the conditions specified in the breakpoint are valid.

Use the Breakpoint Enable (BE) command to reenable a disabled breakpoint.



## BE (Breakpoint Enable)

### Syntax

BE {*breakpoint* [[*breakpoint...*]] | \*}

### Parameters

*breakpoint*

The number of the breakpoint to be enabled. Use the [Breakpoint List \(BL\)](#) command to display currently set breakpoints and their numbers. Specify ranges of breakpoints with a hyphen. Separate multiple breakpoints with spaces or commas.

### Description

Restores one or more breakpoints that were temporarily disabled by the [Breakpoint Disable \(BD\)](#) command. BE\* enables all breakpoints.



## **BL (Breakpoint List)**

### **Syntax**

BL

### **Description**

Lists all breakpoints. For each breakpoint, the command displays the following:

- The breakpoint number
- The breakpoint status, where "E" is for enabled, "D" is for disabled, "V" is for virtual, and "U" is for unknown address. A virtual breakpoint is a breakpoint for code that is not currently loaded.
- The conditional information specifying the breakpoint, such as the address, expression, and length. If a breakpoint has a pass count, the remaining number of times that the breakpoint will be ignored is listed in parentheses.

BL also displays commands to execute, message/message classes (in the case of message breakpoints), thread and process number.

See also : [The Context Operator](#)



## BP (Set Breakpoint)

### Syntax

`[[thread]] BP[[breakpoint]] [[location]] [[condition]] [[option...]]`

### Parameters

#### *thread*

The thread that the breakpoint will apply to. See [Process and Thread Syntax](#).

#### *breakpoint*

The breakpoint number to be set. If the given breakpoint number already exists, the new breakpoint will replace the old.

#### *location*

The memory location of the breakpoint, in the format given in the table below.

#### *condition*

One of the breakpoint conditions given in the table below. You can specify multiple conditions for a breakpoint.

#### *option*

One of the breakpoint options given in the table below. Separate multiple options with spaces.

### Description

Sets a breakpoint. You can combine locations, conditions, and options to set different kinds of breakpoints. If you do not specify a thread, the breakpoint will apply to all threads.

If you want to put a breakpoint on a C++ public, enclose the expression in parentheses. For example, "BP (?? MyPublic)" or "BP (operator new)".

On x86 machines, WinDbg will use debug registers to implement watchpoints if:

- there is a debug register available
- memory size is 1
- memory size is 2 and address is WORD aligned
- memory size is 4 and address is DWORD aligned

You can use the following options when setting a breakpoint:

Location	Description
<code>[[{[[<i>procedure</i>]], [[<i>module</i>]], [[<i>exe</i>]]}]]</code>	The address for the breakpoint.
<code><i>address</i></code>	
<code>[[{[[<i>procedure</i>]], [[<i>module</i>]], [[<i>exe</i>]]}]]</code>	The line number for the breakpoint.
<code>@<i>line</i></code>	

Condition	Description
<code>?<i>expression</i></code>	Break if <i>expression</i> is true.
<code>=<i>address</i> [/R<i>count</i>]]</code>	Break if memory at <i>address</i> has changed. Use the /R option to specify the number of bytes to check (default is 1).

Option	Description
<i>/Pcount</i>	Ignore the breakpoint <i>count</i> times.
<i>/Ccmdlist</i>	Execute <i>cmdlist</i> when the breakpoint is hit. The <i>cmdlist</i> parameter is a semicolon-separated list of one or more debugger commands. If <i>cmdlist</i> includes multiple commands, enclose it in quotes (").
<i>/Mmessagename</i> <i>/Mmessageclass</i>	Break only if the given Windows message or message class has been received.
<i>/Q</i>	Suppress the unresolved-breakpoint dialog box for this breakpoint.
<i>/Hprocess</i>	Specify the process number to attach the breakpoint to. Defaults to all threads in the process if <i>/T</i> is not used.  If <i>/H</i> is not specified and no debuggee is running, the default is process 0.
<i>/Tthread</i>	Specify the thread number to attach the breakpoint to. Defaults to the current process if <i>/H</i> is not used.

See also : [The Context Operator](#)



## C (Compare Memory)

### Syntax

*C range address*

### Parameters

*range*

The first memory area to compare. See [Range Syntax](#).

*address*

The starting address of the second memory area.

### Description

Compares the values held in two memory areas. Specify the first area with the *range* parameter. Specify the starting address of the second area with *address*. The second area is the same length as the first. If the two areas are not identical, WinDbg will display all memory addresses in *range* where they do not agree.



## DA, DB, DC, DD, DI, DS, DT, DU, DW (Display Memory)

### Syntax

D{A|B|C|D|I|S|T|U|W} [[*range*]]

### Parameters

*range*

The memory area to display. See [Range Syntax](#).

### Description

Displays the contents of memory in the given range. Each line shows the address of the first byte in the line, followed by the contents of memory at that and following locations.

If you omit *range*, the command will display memory starting at the ending location of the last Display command. This allows you to continuously scan through memory.

When WinDbg is displaying ANSI or Unicode characters, it will stop displaying characters at the first null byte. When displaying ANSI characters, all characters, including non-printable characters, are displayed using the current code page character set. With Unicode, all non-printable and non-mappable characters are displayed as dots.

Command	Definition	Displays
DA	Display ANSI	ANSI (extended ASCII) characters
DB	Display Bytes (char)	Byte values and ANSI characters
DC	Display Code	Assembly-language instructions (disassembly)
DD	Display Doublewords (long)	Doubleword (four-byte) values and ANSI characters
DI	Display 8-Byte Reals (double)	8-byte hexadecimal values and floating-point representations
DS	Display 4-Byte Reals (float)	4-byte hexadecimal values and floating-point representations
DT	Display 10-Byte Reals (long double)	10-byte hexadecimal values and floating-point representations
DU	Display Unicode	Unicode characters
DW	Display Words (short)	Word values and Unicode characters

### Note

With the DC (Dump Code) command, you can use the standard *range* syntax or *startaddress* I *line* to dump *line* instructions starting at *startaddress*.



## EA, EB, ED, EI, ES, ET, EU, EW (Enter Values)

### Syntax

E{A|B|D|I|S|T|U|W} *address* [[*values*]]

### Parameters

*address*

The starting address to enter values.

*values*

One or more values to enter into memory. Separate multiple numeric values with spaces.

### Description

Enters the values that you specify into memory. If you do not specify any values, the current address and the value at that address will be displayed. You can then enter a new value, preserve the current value in memory by pressing the space bar, or stop entering data by pressing ENTER alone.

When entering numeric values, you can use C-style radix prefixes to override the default radix. Prefix octal constants with a "0o" (for example 0o1776), hexadecimal constants with "0x" (for example 0xF000), and decimal constants with "0t" (for example 0t199).

When entering ANSI or Unicode values, you can include space (" ") characters by enclosing the character string in quotation marks (" or '). If you enclose the string in double quotation marks, WinDbg will automatically null-terminate the string. Single quotation marks (') will not add a null character. You can enter standard C escape characters, such as \t, \007, and \".

Command	Definition	Enter
EA	Enter ANSI	ANSI (extended ASCII) characters
EB	Enter Bytes (char)	Byte values
ED	Enter Doublewords (long)	Doubleword (four-byte) values
EI	Enter 8-Byte Reals (double)	Floating-point numbers
ES	Enter 4-Byte Reals (float)	Floating-point numbers
ET	Enter 10-Byte Reals (long double)	Floating-point numbers
EU	Enter Unicode	Unicode characters
EW	Enter Words (short)	Word values





## **F (Freeze Thread)**

### **Syntax**

`[[thread]] F`

### **Parameters**

*thread*

The thread to be frozen. See [Process and Thread Syntax](#).

### **Description**

Freezes the given thread, causing it to stop and wait until it is unfrozen. Other threads will continue to execute. If no thread is specified, the current thread is frozen.

Use the [Z \(Unfreeze\)](#) command to reenale the thread.



## FIA, FIB, FID, FII, FIS, FIT, FIU, FIW (Fill Memory)

### Syntax

FI{A|B|D|I|S|T|U|W} *range pattern*

### Parameters

*range*

The memory area to fill. See [Range Syntax](#).

*pattern*

One or more values to fill memory with. Separate multiple numeric values with spaces.

### Description

Fills the memory area specified by *range* with *pattern*. The entire *range* will be filled by repeatedly storing *pattern* into memory.

When entering numeric values, you can use C-style radix prefixes to override the default radix. Prefix octal constants with a "0o" (for example 0o1776), hexadecimal constants with "0x" (for example 0xF000), and decimal constants with "0t" (for example 0t199).

When entering ANSI or Unicode values, you can include space (" ") characters by enclosing the character string in quotation marks (" or '). If you enclose the string in double quotation marks, WinDbg will automatically null-terminate the string. Single quotation marks (') will not add a null character. You can enter standard C escape characters, such as \t, \007, and \".

Command	Definition	Fill
FIA	Fill ANSI	ANSI (extended ASCII) characters
FIB	Fill Bytes (char)	Byte values
FID	Fill Doublewords (long)	Doubleword (four-byte) values
FII	Fill 8-Byte Reals (double)	Floating-point numbers
FIS	Fill 4-Byte Reals (float)	Floating-point numbers
FIT	Fill 10-Byte Reals (long double)	Floating-point numbers
FIU	Fill Unicode	Unicode characters
FIW	Fill Words (short)	Word values



## FR (Floating-Point Registers)

### Syntax

```
[[thread]] FR [[register [=value]] ]]
```

### Parameters

*thread*

The thread from which the floating-point registers are to be read. See [Process and Thread Syntax](#).

*register*

The floating-point register to display or modify.

*value*

The floating-point value to assign to the register.

### Description

Displays or modifies floating-point registers. If you do not specify a thread, the current thread is used.

If you do not specify a register, all of the floating-point registers are displayed. If you specify a register, the command displays the current value of the register and prompts for a new value. If you specify both a register and a value, the command changes the register to contain the value.

Use the [R \(Registers\)](#) command to view and modify standard registers.



## G (Go)

### Syntax

`[[process|thread]] G [[breakaddress]]`

### Parameters

*process*

The process to execute. See [Process and Thread Syntax](#).

*thread*

The thread to execute. See [Process and Thread Syntax](#).

*breakaddress*

The address for an unconditional breakpoint.

### Description

Starts executing the given process or thread. Execution will halt at the end of the program, when *breakaddress* is hit, or when another event causes the debugger to stop.

### Note

The breakpoint associated with *breakaddress* will only be hit by the current thread. Other threads that execute the code at that location will not be stopped.



## GH (Go-- Exception Handled)

### Syntax

`[[thread]] GH`

### Parameters

*thread*

The thread to execute. This thread must have been stopped by an exception. See [Process and Thread Syntax](#).

### Description

Marks the given thread's exception as having been handled and allows the thread to restart execution at the instruction that caused the exception. Use the [GN \(Go-- Exception not Handled\)](#) command to continue execution without marking the exception as having been handled.



## GN (Go-- Exception not Handled)

### Syntax

`[[thread]] GN`

### Parameters

*thread*

The thread to execute. This thread must have been stopped by an exception. See [Process and Thread Syntax](#).

### Description

Continues execution of the given thread without marking the exception as having been handled. This allows the debuggee's exception handler to handle the exception. Use the [GH \(Go-- Exception Handled\)](#) command to continue execution after marking the exception as having been handled.



## K, KB, KN, KS, KV (Display Stack Backtrace)

### Syntax

```
[[thread]] K[BNSV] [[framecount]]
```

### Parameters

*thread*

The thread whose stack is to be displayed. See [Process and Thread Syntax](#).

*framecount*

Number of stack frames to display.

### Description

Displays the stack frame of the given thread. Each display line shows the name or address of the procedure called, the arguments used on the call, and the address of the statement that called it. You can use any or all of the options in a single command; for example, K, KB, and KBNSV are valid commands. The following table describes the effect of the options:

Option	Effect
none	The K command without any options displays the basic call stack based on debugging information in the executable. It displays the frame pointer, return address, and function names.
B	The B option causes the K command to additionally display the first three parameters to the functions.
N	The N option causes the K command to additionally display the frame numbers for the calls.
S	The S option causes the K command to additionally display source module and line number information for the calls.
V	The V option causes the K command to additionally display runtime function information.



## **L (Restart Debuggee)**

### **Syntax**

L [*parameters*]

### **Parameters**

*parameters*

Command-line options for the debuggee.

### **Description**

Restarts the debuggee with optional *parameters* supplied as command-line options.





## **LM (List Loaded Modules)**

### **Syntax**

LM [/s [/o]] [/f ] [*modulename*]

### **Parameters**

/s List segmented modules.

/f List flat modules.

/o Sorts segmented modules by selector.

*modulename* Module to list

### **Description**

This command lists the specified loaded modules. If you do not specify a *modulename*, LM will list all loaded modules. If /f and /s are both absent from the command line, LM will assume both; the following two commands are equivalent:

> LM

> LM /f /s

Segmented modules are sorted by module name, then selector, unless you specified /o. Flat modules are sorted by base address.



## **LN (List Nearest Symbols)**

### **Syntax**

LN *address*

### **Parameters**

*address*

The address to search for symbols.

### **Description**

Displays the symbols at or near the given address. You can use this command to help determine what a pointer is pointing to. It also can be useful when looking at a corrupted stack to determine which procedure made a call.



## **M (Move Memory)**

### **Syntax**

M *range address*

### **Parameters**

*range*

The memory area to copy. See [Range Syntax](#).

*address*

The starting address of the destination memory area.

### **Description**

Copies the contents of memory from one location to another. It is legal for *address* to be part of *range*. Overlapping moves are handled correctly.



## N (Set Radix)

### Syntax

N *radix*

### Parameters

*radix*

The default number base used for numeric display and entry. Legal values are 8 (octal), 10 (decimal), and 16 (hexadecimal).

### Description

Sets the default number base to *radix*. You can use C-style radix prefixes to override the default radix.

WinDbg's default is base 16.



## **P (Program Step)**

### **Syntax**

`[[thread]] P [[count]]`

### **Parameters**

*thread*

The thread to step through. See [Process and Thread Syntax](#).

*count*

The number of instructions or source lines to step through before stopping.

### **Description**

Executes the instruction (in ASM mode) or source line (in SRC mode) at the instruction pointer. If WinDbg encounters a CALL instruction or interrupt while stepping, the called subroutine will execute before control returns to the debugger.

Use the [T \(Trace\)](#) command to have WinDbg step through subroutine calls and interrupt-handling routines.



## Q (Quit WinDbg)

### Syntax

Q

### Description

Exits WinDbg. This command is identical to choosing Exit from the File menu.



## R (Registers)

### Syntax

```
[[thread]] R [[register [=value]] ]]
```

### Parameters

*thread*

The thread from which the registers are to be read. See [Process and Thread Syntax](#).

*register*

The register to display or modify.

*value*

The value to assign to the register.

### Description

Displays or modifies registers. If you do not specify a thread, the current thread is used.

If you do not specify a register, all of the registers are displayed. If you specify a register, the command displays the current value of the register. If you specify both a register and a value, the command changes the register to contain the value.

Use the [FR \(Floating-Point Registers\)](#) command to view and modify floating-point registers.



## REMOTE (Start Remote Server)

### Syntax

REMOTE [[*pipename* | STOP]]

### Parameters

*pipename*

The name that you want to use for the remote server pipe.

STOP

Ends a currently active remote server.

### Description

Starts a remote server, which lets you get access to the Command window from another machine on the network. You can then enter Command window commands on the other machine to debug the application remotely.

If you do not specify any parameters to REMOTE, the command displays its connection status and the name of the client.

To connect from another machine, run the following in a cmd shell (the Windows NT command prompt):

```
remote /c hostname pipename
```

where *hostname* is the machine name of the machine that is running WinDbg and *pipename* is the same as above.

You now have access from the remote machine to the WinDbg Command window.





## **RT (Register Display Toggle)**

### **Syntax**

`[[thread]] RT`

### **Parameters**

*thread*

The thread from which the registers are to be read. See [Process and Thread Syntax](#).

### **Description**

Toggles the register display (WinDbg KD only). The default is off.



## S+, S- (Set Source/Assembly Mode)

### Syntax

S+

S-

### Description

Use these commands to switch between Source mode (where the debugger steps line-by-line through the source code) and assembly-language mode (where the debugger steps instruction-by-instruction). The S+ command switches to Source mode, and S- switches to assembly-language mode. The status bar displays whether the current mode is SRC or ASM.



## SA, SB, SD, SI, SS, ST, SU, SW (Search Memory)

### Syntax

S{A|B|D|I|S|T|U|W} *range pattern*

### Parameters

*range*

The memory area to search through. See [Range Syntax](#).

*pattern*

One or more byte values or ANSI or Unicode characters to search for. Separate multiple values with spaces.

### Description

Searches through memory to find a specific byte pattern. If the pattern is found, WinDbg will display the first memory address in *range* where it was found.

When entering numeric values to search for, you can use C-style radix prefixes to override the default radix. Prefix octal constants with a "0o" (for example 0o1776), hexadecimal constants with "0x" (for example 0xF000), and decimal constants with "0t" (for example 0t199).

When entering Unicode values to search for, you can include space (" ") characters by enclosing the character string in quotation marks (" or '). If you enclose the string in double quotation marks, WinDbg will automatically null-terminate the string. Single quotation marks (') will not add a null character. You can enter standard C escape characters, such as \t, \007, and \".

Command	Definition	Fill
SA	Search ANSI	ANSI (extended ASCII) characters
SB	Search Bytes (char)	Byte values
SD	Search Doublewords (long)	Doubleword (four-byte) values
SI	Search 8-Byte Reals (double)	Floating-point numbers
SS	Search 4-Byte Reals (float)	Floating-point numbers
ST	Search 10-Byte Reals (long double)	Floating-point numbers
SU	Search Unicode	Unicode characters
SW	Search Words (short)	Word values



## SEB, SEW (Set Error Break, Set Error Warning)

### Syntax

```
SEB [[level]]  
SEW [[level]]
```

### Parameters

*level*

The error level required to break into the debugger (SEB) or display a message (SEW). One of the following:

Level	Triggered by
0	Nothing. Disables breaks or warnings.
1	Severe errors only.
2	All errors.
3	All warnings and errors.

### Description

The SEB command causes the debugger to break after the debuggee causes a system error or warning of the appropriate *level*. The SEW command causes the debugger to display a message under these conditions. If you do not specify a level, the commands will default to level 1.

The SEB command implies an SEW level equal to or greater than its own level. For example, SEB 3 causes the debugger to give warnings at levels 1-3.

These errors are triggered by the Windows subsystem. They are known as RIP's from Win16 where they usually signal a fatal program error. They are often caused by passing a bad value to an API function.



## SX (Set Exceptions)

### Syntax

```
SX [[exception]]  
SXE exception [[message]] [/Ccmdlist1] [/C2cmdlist2]  
SX{D|N} exception [[message]] [/C2cmdlist2]
```

### Parameters

#### *exception*

The exception number that the command acts upon, in the current radix. If you do not specify an exception, the SX command will display information on all exceptions.

#### *message*

Message to display in the Command window when the exception is trapped.

#### *cmdlist1*

Semicolon-separated list of WinDbg commands to execute when an exception is first raised. The /C option is permitted only with the SXE command. Enclose in quotes if *cmdlist1* includes spaces or semicolons.

#### *cmdlist2*

Semicolon-separated list of WinDbg commands to execute after an exception has not been handled. Enclose in quotes if *cmdlist2* includes spaces or semicolons.

### Description

Controls the behavior of the debugger when trapping exceptions before executing exception-handling code. The debugger always halts before execution returns from the exception handler.

Command	Action
SX	Displays the events that the debugger will halt for.
SXD	Causes the debugger to ignore the specified exception and issue an automatic GN command.
SXE	Causes the debugger to halt at the specified exception.
SXN	Causes the debugger to display a message before the exception is passed to the debuggee, and causes WinDbg to issue a GN command after the message is displayed.

The /C option (allowed only with SXE) tells WinDbg to execute the specified debugger commands on the first chance (before the exception is passed to the debuggee). The /C2 option tells WinDbg to execute the specified debugger commands on the second chance (if the exception is not handled by WinDbg or the application). See your documentation on structured exception handling for more information.

When the debugger stops due to an exception, only the GN (Go-- Exception not Handled) and GH (Go-- Exception Handled) commands can be used to continue execution. The SXD and SXN commands automatically call GN.



## T (Trace)

### Syntax

`[[thread]] T [[count]]`

### Parameters

*thread*

The thread to trace through. See [Process and Thread Syntax](#).

*count*

The number of instructions or source lines (depending on the Source/Assembly mode) to step through before stopping.

### Description

Executes the instruction or source line at the instruction pointer and displays the resulting values of all registers and flags. Use the [P \(Program Step\)](#) command to have WinDbg execute subroutine calls or interrupts without returning control to the debugger.



## U (Unassemble)

### Syntax

U *[[range]]*

### Parameters

*range*

The memory area that contains the instructions to unassemble. See [Range Syntax](#). You can use the standard range syntax or *startaddress*  $\pm$  *line* to dump *line* instructions starting at *startaddress*.

### Description

Displays the instructions of the program being debugged. If you do not specify *range*, the debugger displays the instructions generated from the first eight lines of code at the current address.

The 80286 protected-mode mnemonics cannot be displayed.



## X (Examine Symbols)

### Syntax

X[[options]] [[ { [[procedure]], [[module]], [[executable]] } ]] [[pattern]]

### Parameters

#### *options*

Specifies the scope of the symbol search. These options cannot be separated by spaces and must immediately follow the X (for example, XEM). Use the following letters to specify scope:

C

Search current class.

E

Search entire executable or DLL, except for current module.

F

Search current function.

G

Search all global symbols.

L

Search current code block (lexical scope).

M

Search current module.

P

Search all public symbols.

#### *procedure, module, executable*

The area to search. For example, {function1, srcfile.c, target.dll} will search function1 from srcfile.c in target.dll. This is identical to the breakpoint context operator.

#### *pattern*

A pattern to search for. The ? and \* wildcards are supported.

### Description

Displays the symbols in all contexts that match *pattern*. If you do not specify any options, all except public symbols are searched. If you do not specify a pattern, all symbols will be displayed. The search will be case-sensitive if the Ignore Case option is not checked in the Debug command from the Options menu.





## **Z (Unfreeze Thread)**

### **Syntax**

`[[thread]] Z`

### **Parameters**

*thread*

The thread or threads to be unfrozen. See [Process and Thread Syntax](#).

### **Description**

Unfreezes the specified thread. If you do not specify the thread, this command unfreezes the thread that caused the most recent exception, if any. You can freeze threads with the [F \(Freeze\)](#) command.



## **Using the Editor**

WinDbg includes an editor, with which you can view and modify text files from within the debugger.

Basic Operations with Files

Editing Tasks

Moving Through a File

Controlling Your Windows



## **Basic Operations with Files**

Opening a File

Creating and Saving a New File

Merging Files

Saving a File

Saving All Open Files

Saving a File Under a New Name

Making a Source Window Read Only

Closing the Active Window

Exiting WinDbg



## **Editing Tasks**

[Cutting and Pasting Text](#)

[Copying Text](#)

[Copying Text Between Applications](#)

[Copying the Contents of the Help Window](#)

[Deleting Text](#)

[Finding Text](#)

[Replacing Text](#)

[Using Regular Expressions](#)

[Going to a Line](#)

[Undoing and Redoing an Action](#)

[Setting Tabs](#)



## **Controlling Your Windows**

[Creating a New Window for a Source File](#)

[Arranging Windows](#)

[Minimizing Windows within WinDbg](#)

[Increasing the Size of the Edit Window](#)

[Changing Display Colors](#)

[Setting the Font and Font Size](#)

[Making a Source Window Read Only](#)

[Using Workspaces](#)



## **Moving Through a File**

Setting a Tag in a Source Window

Removing a Tag or All Tags in a File

Moving to Tags in a Source File



## Opening a File

Use the Open command from the File menu (or press CTRL+F12) to load an existing text file.

To open a file:

1. From the File menu, choose Open.
2. If the file is on a network share, click the Network button to select the share.
3. Choose a drive from the Drives list box. The default drive is the current drive.
4. From the Directories list box, double-click a directory where the file is stored (or down a directory path to the directory where the file is stored.)
5. Choose a file type in the List Files of Type box. Only files with the chosen extension(s) are displayed in the File Name box.
6. Double-click a filename, or click a filename and choose OK.

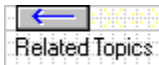
The names of the four most recently opened files are displayed at the end of the File menu. To open one of these files, choose its name from the menu.

To open a file as read-only, check the Read Only box in the Open File dialog box. To make a file read-only after opening it, check the Read Only command in the Edit menu. If you open a read-only file, the Read Only command is automatically checked. To edit a version of the file, uncheck the Read Only command, make your edits, and then save the file under a different name (Save As command).

You can use wildcard patterns in the File Name text box to display file types. For example, entering "\*.INC" displays files only with the .INC extension. The new wildcard pattern is retained in a session until you change it. You can use any combination of wildcard patterns, separated by semicolons. For example, entering "\*.INC; \*.H; \*.CPP" displays all files with these extensions.

### Note:

The maximum number of characters in a line is 251.



Related Topics



### Creating and Saving a New File

Use the New command from the File menu to open a blank source window for a new file. This does not affect other text files you may have open.

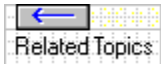
To create a new file:

1. From the File menu, choose New.
2. Enter text in the new window.
3. From the File menu, choose Save As (or press F12).
4. In the Save As dialog box, select the directory where you want the file stored.
5. Type a filename in the File Name box, then choose OK.

If you open new source windows without closing or saving them, WinDbg temporarily names them UNTITLED 1 through UNTITLED *n*.

#### Note:

The maximum number of characters in a line is 251.







### Merging Files

Use the Merge command from the File menu to insert another file at the insertion point. Merge replaces any selected text in the active source window with the text of the other file.

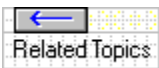
To merge files:

1. Either place the insertion point where you want to merge text, or select the text that you want to be replaced.
2. From the File menu, choose Merge.
3. Select the file that you want to merge into the active window:
  - a. If the file is on a network share, click the Network button to select the share.
  - b. Select the appropriate drive in the Drives box.
  - c. Select the appropriate directory in the Directories list box.
  - d. Select the file. If you don't see the file listed, make sure you have selected the correct file type in the List Files of Type box.
4. Choose OK.

As an alternative to this procedure, you can open a file with the editor and copy and paste text from the Clipboard.

#### Note:

The maximum number of characters in a line is 251.



Related Topics



### **Saving a File**

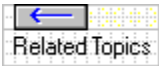
Use the Save, Save As, or Save All command from the File menu to save the contents of the active source window under the name shown in the title bar.

If your file is untitled, the Save As dialog box appears. You can also select Save As to save your file under a different name. In this case, the original file does not change.

The Save All command saves all changed source files.

#### **Note:**

The maximum number of characters in a line is 251.





### **Saving a File Under a New Name**

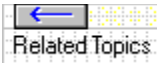
Use the Save As command from the File menu (or press F12) to save a newly created file or copy an existing file to another name.

Procedure:

1. Make sure that the active window is the source window for the file that you want to save.
2. Choose Save As from the File menu.
3. Specify the name of the file that you want to save:
  - a. If the file is on a network share, click the Network button to select the share.
  - b. Select the appropriate drive in the Drives box.
  - c. Select the appropriate directory in the Directories list box.
  - d. Type the filename in the File Name box.
4. Choose OK.

#### **Note:**

The maximum number of characters in a line is 251.



Related Topics



### **Saving All Open Files**

Use the Save All command from the File menu to save all changed or new files with one command.

#### **Note:**

The maximum number of characters in a line is 251.



Related Topics



### **Making a Source Window Read Only**

To avoid making inadvertent changes to the current file, check the Read Only command from the Edit menu. A check mark next to the menu command and READ in the Status Bar indicate that a source file is read-only. If you have already modified the file, you cannot mark it as read-only. To reenale writing to the file, uncheck the Read Only command.

If you open a read-only file, the Read Only command is automatically checked. To edit a version of the file, uncheck the Read Only command, make your edits, and then use the Save As command from the File menu to save the file under a different name.

If the debugger opens a file for source line tracking, the file will be opened in read-only mode.

When a source window to a file is restored as part of a workspace, the read-only attribute is also restored.





### **Closing the Active Window**

You can close the active window in several ways:

1. Use the Close command from the File menu. This will close the active window and any additional windows to the same file.
2. Double-click on the control-box menu for the window. This will close only the active window.
3. Click on Close from the control-box menu, or press CTRL+F4.





## Exiting WinDbg

Use the Exit command from the File menu (or press ALT+F4) to quit WinDbg. You can also use the **Q** command from the Command window.

If you are still debugging a program, WinDbg will ask you if you want to exit. If there are unsaved files open, WinDbg will ask you if you want to save them.





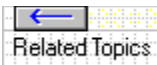
### **Cutting and Pasting Text**

Use the Cut command from the Edit menu to remove the selected text from the active window and place it in the Clipboard. Use the Paste command from the Edit menu to paste the text at the insertion point. You can also press SHIFT+DEL to cut highlighted text to the Clipboard and press SHIFT+INS to paste text from the Clipboard to the insertion point.

You can also use the Quickedit feature: Highlight the text that you want to copy by holding down the left mouse button and dragging the mouse pointer. Click the right mouse button to copy the highlighted text to the Clipboard. Move the mouse pointer to where you want to paste the text and click the right button again. You can use the same technique to copy text from other windows to the Source window.

#### **Note:**

The maximum number of characters in a line is 251.



Related Topics





### Copying Text

Use the Copy command from the Edit menu to copy the selected text from the active window and place it in the Clipboard. Use the Paste command from the Edit menu to paste the text at the insertion point.

You can also use the Quickedit feature: Highlight the text that you want to copy by holding down the left mouse button and dragging the mouse pointer. Click the right mouse button to copy the highlighted text to the Clipboard. Move the mouse pointer to where you want to paste the text and click the right button again. You can use the same technique to copy text from other windows to the Source window.





### **Copying Text Between Applications**

Use the Paste command from the Edit menu to paste text that you have copied to the Clipboard in any standard Windows application. All such applications have a Copy command that copies text to the Clipboard.





### **Copying the Contents of the Help Window**

By cutting and pasting, you can copy information from the Help window to the Clipboard and then to a source window or another application:

1. From the Edit menu of the Help window, choose Copy.
2. Select the text to copy from the Copy dialog box.
3. Click the Copy button to copy the text to the Clipboard.
4. Switch to a source window or another Windows-based application.
5. Choose the Paste command from the Edit menu.

To copy the contents of the entire Help window to the clipboard without displaying the dialog box, press CTRL+INS.





### **Deleting Text**

Use the Delete command from the Edit menu or press DEL to delete selected text. The deleted text is not placed in the Clipboard and cannot be pasted. Press SHIFT+DEL to delete text to the Clipboard.



<b>Option or Command</b>	<b>Description</b>
Match Whole Word Only option	Matches all occurrences of the Find What string that are not preceded or followed by an alphanumeric character or the underscore (_).
Match Case option	Searches for text that matches the capitalization of the text string.
Regular Expression option	Uses special characters to search for text.
Direction Up option	Searches from the insertion point to the beginning of the file.
Direction Down option	Searches from the insertion point to the end of the file.
Find Next command	Finds the next occurrence of the text string.
Tag All command	Marks each line the text occurs on. Use the Tag commands in the View menu to move to each occurrence

If your insertion point is on a word in a line, the word is displayed as the search text in the Find What text box. If the insertion point is between words, the editor displays the word to the right of the insertion point. If there is no word to the right of the insertion point, the editor displays the word to the left. Failing that, the editor displays the text from the previous search.



## Replacing Text

Use the Replace command from the Edit menu to search the active window for a text string and replace it with another text string. You can specify:

- A whole-word match
- A case-sensitive match
- A general pattern search using a regular expression

To replace text:

1. Position the insertion point where you want to start the find-and-replace procedure.  
The editor uses the location of the insertion point to select a default search string to replace. For this procedure, assume that the Find What text box is blank.
2. From the Edit menu, choose Replace.
3. In the Find What text box, type the search text or regular expression.
4. In the Replace With text box, type the replacement text. You cannot use regular expressions as replacement text.
5. Select any Replace options. If you choose Find Next, an abbreviated version of the Replace dialog box is displayed. This dialog box lets you choose the Find Next, Replace, or Replace All commands.

The drop-down list box connected to the Find What text box records the last 16 text items you searched for. Click the drop-down arrow to the right of the text box or press ALT+DOWN-ARROW to open the list. To begin replacing text, select a text item from the list, then choose Find Next or Replace All.



<b>Option or Command</b>	<b>Description</b>
Match Whole Word Only option	Matches all occurrences of the Find What string that are not preceded or followed by an alphanumeric character or the underscore (_).
Match Case option	Replaces text that matches the capitalization of the text string.
Regular Expression option	Uses special characters to define the Find What search string.
Find Next command	Finds the next occurrence of the text string w/o replace.
Replace command	Replaces the text string and finds the next one. This command is available only after you choose the Find Next command.
Replace All command	Replaces all occurrences of the text string.





## Using Regular Expressions

A regular expression is a search string that uses special characters to match a text pattern in a file.

You can use the following special characters in regular expressions:

Character	Use
. (period)	Matches any single character. <a href="#">Example</a>
^ (caret)	Matches at the beginning of a line. <a href="#">Example</a>
\$ (dollar sign)	Matches at the end of a line. <a href="#">Example</a>
* (asterisk)	Matches zero or more occurrences of the character(s) preceding the asterisk. <a href="#">Example</a>
+ (plus sign)	Matches one or more occurrences of the character(s) preceding the plus sign. <a href="#">Example</a>
[ ] (brackets)	Matches sets of the characters specified within the brackets. <a href="#">Example</a>
\ (backslash)	Interprets the next character as a literal character---with no special meaning. <a href="#">Example</a>





### **Undoing and Redoing an Action**

Use the Undo command from the Edit menu to reverse previous editing actions, including replace actions. Use the Environment command from the Options menu to set the size of the Undo buffer.

After an Undo, use the Redo command from the Edit menu to reinstate the action that you just reversed with Undo.





### **Setting Tabs**

Use the Environment command from the Options menu to set the following Tab settings:

- Tab Stops. Enter the number of spaces that you want a tab to be equivalent to.
- Keep Tabs. Choose this option if you want tab characters to be retained.
- Insert Spaces. Choose this option if you want tab characters to be converted to spaces.





## Arranging Windows

WinDbg can arrange your windows in several patterns.

- For an overlapped pattern, choose Cascade from the Window menu.
- For a side-by-side pattern, choose Tile from the Window menu.
- For a CodeView-like pattern, choose Arrange from the Window menu.
- To arrange iconized windows, choose Arrange Icons from the Window menu.

You can size windows by using the standard Windows techniques. When you save your workspace, the size and arrangement of the windows is saved.





### **New Window**

The New Window command opens a new window to the source file. This command is valid only for source windows.

If Overlay Source is checked, then any new source windows that you open will appear in the same location at which the first source window opened.





### **Minimizing Windows within WinDbg**

You can click on the minimize button to minimize most windows within WinDbg. The Help window minimizes on the desktop. To restore a minimized window, double-click on its icon.





### **Increasing the Size of the Edit Window**

You can remove the Status Bar, Toolbar, and Scroll bars to increase the size of your working area within WinDbg.

Check or uncheck the Toolbar and Status Bar commands in the View menu to display or remove the Toolbar and Status Bar.

Use the Environment command from the Options menu to control the display of the Scroll Bars. You can remove one or both scroll bars.





### **Setting a Tag in a Source Window**

You can tag frequently accessed lines in your source file and then use the Next Tag and Previous Tag commands from the View menu to move to the tagged lines.

Use the Toggle Tag command from the View menu to tag the current line in the source window. Use the Toggle Tag command again to clear the tag. Use the Clear All Tags command from the View menu to clear all tags in the source window.

To tag every line that contains a particular string, use the Find command from the Edit menu to search for the string and click on the Tag All button in the Find dialog.







### **Removing a Tag or All Tags in a File**

Use the Toggle Tag command from the View menu to clear the tag on the current line in the source window. Use the Clear All Tags command from the View menu to clear all tags.





### **Moving to Tags in a Source File**

Use the Next Tag or Previous Tag commands from the View menu to move to the next or previous tagged line.





## Finding Text

Use the Find command from the Edit menu to search the active window for a text string. You can specify:

- A whole-word match
- A case-sensitive match
- A general pattern search using regular expressions

To find a text string:

1. Position the insertion point where you want to start the find procedure.

The editor uses the location of the insertion point to select a default search string. For this procedure, assume that the Find What text box is blank.

2. From the Edit menu, choose Find.
3. In the Find What text box, type the search text or regular expression.
4. Select any of the Find options. If you choose Find Next, an abbreviated version of the Find dialog box is displayed. This dialog box lets you choose the Find Next or Tag All commands.

The drop-down list box connected to the Find What text box records the last 16 text items you searched for. Click the drop-down arrow to the right of the text box or press ALT+DOWN-ARROW to open the list. To begin a search, select a text item from the list, then choose Find Next or Tag All.

Use CTRL+] to find the match to a parenthesis, brace, or bracket in your source code. Place the insertion point in front of the parenthesis, brace, or bracket and press CTRL+]. The insertion point moves forward or backward to the matching item. Press CTRL+] again to return the insertion point. If a match cannot be found, the editor beeps.





### The Find Dialog Box

Command	Description
Find Next	Finds the next occurrence of the search string.
Tag All	Tags all occurrences of the search string.





## The Replace Dialog Box

### Command

### Description

Find Next

Finds the next occurrence of the text string without replacing it.

Replace

Replaces the text string and finds the next one.

Replace All

Replaces remaining occurrences of the text string.





### **Moving to a Specific Line**

Use the Goto Line command from the View menu to go to a specific line number in the source file. Enter the line number in the dialog box.





A dynamic-link library (DLL) is a file containing routines accessible to an executable (EXE) file. Routines in a dynamic-link library are not added to the EXE file when the EXE is built. Instead, the DLL is loaded into memory when the EXE is run. The EXE can then call routines in the DLL.



An EXE (executable) file is a set of instructions a computer can read. A compiler takes your program source code and translates it into machine code.

An .OBJ file contains an intermediate code produced by compiling a program and used by the linker to form an executable file.

The active window is the window where all actions take place. The title of the active window is highlighted to distinguish it from the other visible windows. To make a window active, click the window or choose the window title from the Window menu.

A breakpoint is the location where program execution will halt. Breakpoints can be set unconditionally at a location or can be made conditional on the basis of expressions or messages.

Selected text is any highlighted block of text. You can select text with the mouse or text-selection keys. Selected text can be copied, cut, deleted, replaced, or printed.

A function is a group of statements that performs a specific task and often returns a value to the statement that calls it. C functions allow you to write well-organized programs that perform different tasks in separate parts.

A constant is a value that does not change during program execution, including literal and number strings and symbolic constants.

A structure is a set of elements, which may be of different types, grouped under a single name.



An argument is a value passed to a function.

A search for text that matches the capitalization of the text string

The icon (symbolized by a spacebar) that appears in the upper-left corner of a window. The Control-menu box opens the Control menu for the window.

When a window is maximized in WinDbg, the Control-menu box for the window appears as the first item in the menu bar for WinDbg.

When color-coding is turned on, language keywords, identifiers, comments, and strings are coded in different colors for easier readability of your program source code.

A default extension is an extension determined by the development environment.

The Debug option creates files with debugging information for the development environment debugger or CodeView. The Debug option is the default.

The Errors window displays only error output.

An expression is a combination of operands and operators that yields a value.



The Tag All command tags all lines that contain the search text.

A help topic is a screen of information reached by a link.

An icon is a graphical link to a help topic.



The Match Case option indicates that only an exact, case-sensitive match will be allowed. This option appears in the Find and Replace dialog boxes.

The minimize button is the button with the down arrow at the right end of the title bar

A multimodule program contains more than one component file, including source files (.C ), header files (.H), object files (.OBJ), and library files (.LIB), a module-definition file (.DEF) and a resource script (.RC).

UNTITLED1 through UNTITLEDnn

A pop up is a window displayed on top of the current window. A pop up is dismissed by clicking anywhere on your screen.



Once a window is made read-only, all of the editing commands---Undo, Redo, Cut, Copy, Paste, and Clear---are unavailable and text entry is not permitted.

The Release option creates files without debugging information. Release versions are typically smaller and may execute more quickly.

A resource is read-only data stored in your application's EXE file or DLL. Windows reads a resource from disk on demand. Certain types of resources have prescribed formats recognized by Windows. These include bitmaps, icons, cursors, dialog boxes, and fonts.

A single-module program contains just one C source file.

A number in the Length text box specifies the size of a memory location. The number should always be a positive.

A source file is an ASCII file, such as a program or header file, stored on your disk. WinDbg displays your source files in a source window.

A source window contains any editable file, such as a source or header file.

The Status Bar is displayed at the bottom of the WinDbg desktop window. It displays the information about commands, source/ assembly mode, process number, thread number, overtype mode status, WordStar key sequence status, read-only status, column number, and line number.



A tag is an internal "bookmark" maintained by the development environment. A tagged line is highlighted. The Next Tag and Previous Tag commands in the View menu can be used to move between tags.

A text string consists of ordinary, readable characters, including uppercase and lowercase letters of the alphabet, the numerals 0 through 9, and punctuation marks.

The Match Whole Word Only option matches all occurrences of the Find What string that are not preceded or followed by an alphanumeric character or the underscore (\_).

For example, "foo" finds "int foo;" and "\* foo.next;", but not "printf( "foolish" );" or "my\_foo = 3;".

A variable is an identifier that the compiler assigns to a memory location. Programs can temporarily store values in memory by assigning the values to variables.

A regular expression is a search string that uses special characters to match a text pattern in a file. Unlike a regular expression, a literal search string finds a literal match with itself (the search string "procedure" finds the occurrences of the word "procedure"). Regular expressions allow you to go beyond literal searches. You might use regular expressions to search for all five-digit numbers in your source file or for all strings in quotation marks.

Global means the broadest possible scope, in which a variable or constant definition is recognized by the entire application.

A union is a set of values of different types that occupy the same storage space.

**Character:** Period .

**Matches:** Any single character

**Example:** Log..

The example above finds any occurrence of "log" with two additional literal characters, such as "logic" or "log10". The text pattern can occur in the middle of a word: log.. would find "logar" in logarithm, for instance.



**Character:**   Caret ^

**Matches:**    At the beginning of a line

**Example:**     ^int

The example above finds "int" at the beginning of any line. For example,

int matherr( struct exception \*except).

Since spaces are considered characters, int would not be found if indented by spaces or a tab.

**Character:** Dollar sign \$

**Matches:** At the end of a line

**Example:** program\$

The example above finds "program" at the end of any line. You may want to combine special characters to increase the scope of the search:

program...\$

This example finds "program. }". A search with the regular expression in the first example would have passed over this construction.

**Character:** Asterisk \*

**Matches:** Zero or more occurrences of the character(s)  
preceding the asterisk

**Examples:** to\*t\*, and bu[gnr]\*y

The first example matches "t", "tt", "ttt", "tot", "tott", "too", and "toot".

The second example matches "bu" followed by zero or more occurrences of g, n, and/or r, followed by y: Examples are "buy", "bury", "bunny", and "bungy".

**Character:** Plus sign +

**Matches:** One or more occurrences of the character(s)  
preceding the plus sign

**Examples:** to+t, and [aeiou]+

The first example matches "tot" and "toot" but not "tt".

The second example matches one or more occurrences of "a", "e", "i", "o", and/or "u", such as "a", "uae", and "eieiooo".

**Character:** Brackets [ ]

**Matches:** Sets of characters specified within the brackets

**Example:** x[-+/\*]y

In the example above, the regular expression matches x+y, x-y, x/y, or x\*y, but not x=y or xzy. The special characters plus (+) and asterisk (\*) do not function within the brackets.

The following special characters may be used inside brackets:

Character	Use
Caret(^)	Matches any character except those specified within the brackets. Must be the first character within brackets to be treated as a special character.
Hypen(-)	Matches characters in ASCII order between the characters on either side of the dash, including the delimiting characters. Example: [0-9]+ Example matches: Any digit sequence.

**Character:** Backslash \

**Matches:** Interprets the next character literally

**Example:** `x\*y`

The example above finds "x\*y". Characters preceded by a backslash lose their special meanings and are treated literally.

CTRL+F12

Undo ALT+BACKSPACE

Redo CTRL+BACKSPACE



ALT+F4

CTRL+F4

CTRL+INS

DEL

End of a file CTRL+END

Beginning of a file CTRL+HOME

SHIFT+DEL

Find command	SHIFT+F3 or CTRL+Q F
Repeat last find	F3
Find selected text	CTRL+F3

SaveSHIFT+F12

Save As F12



Save SHIFT+F12

Save As F12

Save SHIFT+F12

Save As F12

Next Error	F4
Previous Error	SHIFT+F4


SHIFT+F5

F5




CTRL+SHIFT+F12

Keyboard F8

Toolbar 




Keyboard F10


Toolbar 

SHIFT+INS

Keyboard      SHIFT+ F9

Toolbar      

Keyboard F9

Toolbar 

CTRL+F9

Cut  
Paste

SHIFT+DEL  
SHIFT+INS

CopyCTRL+INS

PasteSHIFT+INS

Toggle Tag CTRL+F2  
Next Tag F2  
Previous Tag SHIFT+F2



Replace

CTRL+Q A

<b>Action</b>	<b>Shortcut Key(s)</b>
Make Font control active	ALT+N
Make Pts control active	ALT+S
Move between Font and Pts controls once both active	TAB
Move through Font and Pts selections	ARROW keys, HOME, END, PGUP, PGDN
Confirm selection	ENTER
Cancel selection	ESC

1. At the ARC firmware prompt, select Run Setup.
2. From the first menu, select Manage Startup.
3. From the second menu select Change a Boot Selection.
4. Edit the OSLOADOPTIONS variable:  
OSLOADOPTIONS=*option1 option2 ...*

For example:

OSLOADOPTIONS=DEBUG

BOOT.INI is located in the root directory of the boot partition (usually c:\). It is a read-only, hidden, system file. In the File Manager, use the Properties command from the File menu, or use the following attrib command to change the file's attributes so that you can modify it.

```
attrib -r -h -s boot.ini
```

The [operating systems] section of BOOT.INI lists the operating systems that the machine can boot. Add the desired options to a Windows NT line:

```
multi(0)disk(0)rdisk(0)partition(1)\NT="Windows NT" /option1 /option2 ...
```

For example:

```
multi(0)disk(0)rdisk(0)partition(1)\NT="Windows NT" /DEBUG
```





## **Using Help**

WinDbg Online Help is an online reference on the WinDbg debugger for Microsoft Windows NT. It also allows access to information on the Win32 API. Information in Help is organized into [help topics](#).

[The Help Window](#)

[Getting Help](#)

[Creating a Bookmark at a Help Topic](#)

[Annotating a Help Topic](#)

[Copying the Contents of the Help Window](#)

[What are Links, Icons, Pop Ups, and Index Screens?](#)



### **Help Contents**

Use the Contents command from the Help menu to display the top-level Help screen for WinDbg Online Help. Click on an icon in the screen to see a table of contents for the icon's topic.



## Getting Help

- ▶ You can get help in several ways:
  - From the Help Contents screen.
  - From the Help Search function.
  - By pressing F1 while a menu command is highlighted or a dialog box is active.
  - By choosing the Help button in a dialog box.





### **Searching for Help Topics**

The Search command in the Help window or menu is a shortcut to find help topics. The Search command opens a dialog box where you can begin a search. Help topics are indexed by keywords---for instance, the help keyword "open".

- ▶ To search for help topics under the keyword "open":
  1. Choose the Search button in the Help window (or Search from the Help menu)
  2. Type "open" in the text box or choose a term from the list.
  3. Choose Show Topics. All of the topics under "open" are listed.
  4. Select a topic and then choose the Go To button.



### **Creating a Bookmark at a Help Topic**

Just as you can place bookmarks in a book to mark specific references, you can place bookmarks at Help topics you frequently refer to.

- ▶ To place a bookmark at the current topic:
  1. From the Bookmark menu in Help, choose Define.
  2. Type a name in the Bookmark Name box and choose OK. The bookmark name now appears on the Bookmark menu in Help.
- ▶ To go to a bookmark:
  1. Select the Bookmark menu in Help.
  2. Choose the topic (bookmark name) you want to view.

Underlined numbers precede the first nine bookmark titles. You can press the corresponding number key to go to a marked topic. If more than nine bookmarks have been defined, choose More from the Bookmark menu to see them.

- ▶ To remove a bookmark:
  1. From the Bookmark menu in Help, choose Define.
  2. Select the bookmark you want to remove.
  3. Choose Delete.



### **Annotating a Help Topic**

You can add your own comments to a help topic. Adding comments to a Help topic is called "annotating."

- ▶ To annotate the current Help topic:
  1. From the Edit menu in Help, choose Annotate.
  2. Type the text you want to add in the box provided in the Annotate dialog box. (If you make a mistake, press Backspace to delete text.) Text wraps automatically in the edit box, but you can end a line before it wraps by pressing Ctrl+Enter.
  3. Choose Save when finished.
- ▶ To view an annotation:
  1. Go to the topic where you made the annotation. When you make an annotation, Help places a paper clip icon to the left of the topic title.
  2. Click the paper clip icon or press Tab to highlight the paper clip icon, then press Enter.
  3. Choose Cancel when you have finished reading the annotation.
- ▶ To remove an annotation:
  1. Go to the topic where you made the annotation.
  2. Click the paper clip icon or press Tab to highlight the paper clip icon, then press Enter.
  3. Choose Delete.



### **Copying the Contents of the Help Window**

By cutting and pasting, you can copy information in the Help window through the Clipboard to a source window or another application.

- ▶ To copy text to the Clipboard:
  1. From the Edit menu of the Help window, choose Copy.
  2. Select the text to copy from the Copy dialog box.
  3. Choose Copy to copy the text to the Clipboard.
  4. Switch to a source window or another Windows application.
  5. Choose the Paste command from the Edit menu.

To copy the contents of the entire Help window to the clipboard without displaying the dialog box, press Ctrl+Ins.



### **The Help Window**

The Help window works like other windows. The window can be moved, resized, scrolled, and closed with the standard Windows menu and keyboard commands. To exit Help, choose Exit from the File menu in the Help window.

The configuration of the Help window is not saved in a Workspace. When you quit Help, however, the size and location of the Help window is saved in the registry. This means you can size and place the Help window to fit your programming needs; when you open Help, the window is then displayed as customized.



### **Key to the Buttons in the Help Window**

A series of buttons are displayed at the top of the Help window. The commands in the button bar help you navigate through help.

<b>Command</b>	<b>Description</b>
Contents	Displays the table of contents for help topics.
Search	Lists all the keywords in Help. By typing or selecting a keyword, you can search for and go to specific Help topics.
Back	Displays the previous help topic.
History	Displays a list of the last 40 opened help topics viewed. Double-click a title to display that help topic.



### **Links, Icons, Pop Ups, and Index Screens**

A link is a connection to another help topic. Links are identified by a solid underline. When you click a link (or press Tab to the link and press Enter), the Help window displays the help topic called by the link name.

Icons can be graphical links. Click on the following icon for an example:



A pop up is a link with a dotted underline. The help topic for a pop up is displayed in a small pop-up window. Pop ups are often, but not always, definitions of terms.

Index screens are listings of help topics for a category. They are usually pop ups. An arrow pointing left at the bottom of a help topic takes you back to the index screen for related topics.



### **Example of a Link**

Click the word [return](#) to go back to the previous screen.





### **Example of an Icon Link**

Click the icon in the title above to go back to the previous screen.



### **Example of an Index Screen**

Book a Flight to a Distant Star

Lions on the Plain

Speaking of LISP

Moorish Influence on Spanish Architecture





## **Quick Look**

[What You Need to Know About Help](#)

[Shortcut Keys for Help Information](#)

[Key to the Toolbar](#)

[Key to the Status Bar](#)

[Shortcut Keys for Debugging](#)

[Shortcut Keys to Move the Cursor](#)

[Shortcut Keys to Delete Text](#)

[Shortcut Keys to Insert and Copy Text](#)

[Shortcut Keys to Find and Replace Text](#)

[Shortcut Keys to Scroll Text](#)

[Shortcut Keys to Select Text](#)



### **What You Need to Know about Help**

A help topic is one or more screens of information. Help topics are connected by links. To display another help topic, you activate a link.

You activate a link by clicking on a mouse-sensitive hot spot in a piece of text or graphics. The mouse pointer changes shape when it is on a hot spot. You can also press TAB to move to the next hot spot and then press ENTER to activate the link.

A jump has a solid underline. Click on a jump to switch the Help window to the topic pointed to by the jump. Click on Back in the toolbar to return to the original topic.

Text that has a dotted underline is linked to a pop-up window, which usually contains the definition of the underlined term or a list of information. Click on the text to display the pop-up window. Click the mouse again to dismiss the pop-up and return to the main window.

Icons can also provide links. Click on an icon to either jump to another topic or display a pop-up window.



You can get help in several ways:

- Choose the Help button in a dialog box.
- Choose the Contents or Search command from the Help menu.
- Press F1 while a menu command is highlighted.



## **Shortcut Keys for Help Information**

### **Help**

### **Key**

Get help on the current  
dialog or menu

F1

Move to next hot spot

TAB

Move to previous hot spot

SHIFT+TAB



## Key to the Toolbar



Click on an icon for more information.



Equivalent to the Go command from the Run menu.



Equivalent to the Halt command from the Run menu.



Sets a breakpoint at the current line.



Equivalent to the Quickwatch command from the Debug menu.



Equivalent to the Trace Into command from the Run menu.



Equivalent to the Step Over command from the Run menu.



Turns on source-mode debugging. Equivalent to switching to source mode with the Toggle Source/Asm Mode command from the Run menu.



Turns on assembly-mode debugging. Equivalent to switching to assembly mode with the Toggle Source/Asm Mode command from the Run menu.



Opens the Options dialog, if any, for the current window.



## Key to the Status Bar

Contents	Description
Message box	Displays messages from the environment.
SRC/ASM	Indicates whether WinDbg is in source or assembly mode.
pid	Shows the ID number of the process being debugged.
tid	Shows the ID number of the thread being debugged.
^Q	Indicates a WordStar keystroke sequence.
OVR	Indicates overtype mode.
READ	Indicates read-only status.
CAPS	Indicates that CAPS LOCK is on.
NUM	Indicates that NUM LOCK is on.
Line	Displays the line number at the insertion point in the current window.
Column	Displays the column number at the insertion point in the current window.





## Shortcut Keys for Debugging

### Debug Action

### Key

Help.	F1
Start program execution from beginning.	SHIFT+F5
Terminate a running application.	ALT+F5
Continue program execution from current statement.	F5
Attach to a process.	F6
Execute program to current cursor position.	F7
Execute next program statement as a single step.	F8
Set a breakpoint on the current line.	F9
Quickwatch.	SHIFT+F9
Single step, tracing around a function call.	F10



### Shortcut Keys to Move the Cursor

Use either the Microsoft or WordStar keystrokes.

Move Cursor	Microsoft	WordStar
One character left	LEFT	CTRL+S
One character right	RIGHT	CTRL+D
Word left	CTRL+LEFT	CTRL+A
Word right	CTRL+RIGHT	CTRL+F
Line up	UP	CTRL+E
Line down	DOWN	CTRL+X
First indent level of current line	HOME	
Beginning of current line	HOME+HOME	CTRL+Q S
End of line	END	CTRL+Q D
Top of window		CTRL+Q E
Bottom of window		CTRL+Q X
Beginning of file	CTRL+HOME	CTRL+Q R
End of file	CTRL+END	CTRL+Q C
Matching brace, bracket, or parenthesis	CTRL+]	



In the Command window, use the up-arrow and down-arrow keys to get access to the command history; use CTRL-UP to move the text cursor off of the input line.



### **Shortcut Keys to Delete Text**

Use either the Microsoft or WordStar keystrokes.

<b>Delete</b>	<b>Microsoft</b>	<b>WordStar</b>
Character at cursor	DEL	CTRL+G
Character to left of cursor	BACKSPACE	CTRL+H
Current line, but copy to Clipboard		CTRL+Y
Rest of word at cursor location		CTRL+T
Selected text	DEL	CTRL+G
From cursor to end of line but copy to the Clipboard		CTRL+Q Y
Selection and copy to the Clipboard	SHIFT+DEL	
Last edit	ALT+BACKSPACE	
Unindent selected lines	SHIFT+TAB	



### Shortcut Keys to Insert and Copy Text

Use either the Microsoft or WordStar keystrokes.

Insert/Copy	Microsoft	WordStar
Keyboard insert mode on or off	INS	CTRL+V
Copy selected text to Clipboard	CTRL+INS	
Copy selected text to Clipboard, deleting it from file	SHIFT+DEL	
Copy current line to Clipboard, deleting it from file		CTRL+Y
Copy to end of line to Clipboard, deleting it from file		CTRL+Q Y
Insert contents of Clipboard	SHIFT+INS	
Insert line break and move the cursor to next line	ENTER	ENTER
Insert line break without moving cursor		CTRL+N
Undo last edit	ALT+BACKSPACE	



There is also a Quickedit feature that lets you copy text from a line in any window and paste it to the end of the Command-window prompt line. To copy text from a line, highlight the text and click the right mouse button. To paste the text to the end of the Command-window prompt line, move the mouse pointer to the Command window and click the right mouse button again.



### **Shortcut Keys to Scroll Text**

Use either the Microsoft or WordStar keystrokes.

<b>Scroll</b>	<b>Microsoft</b>	<b>WordStar</b>
Line up	CTRL+UP	CTRL+W
Line down	CTRL+DOWN	CTRL+Z
Page up	PGUP	CTRL+R
Page down	PGDN	CTRL+C
Left one window	CTRL+PGUP	
Right one window	CTRL+PGDN	
Beginning of file	CTRL+HOME	CTRL+Q R
End of file	CTRL+END	CTRL+Q C



### **Shortcut Keys to Select Text**

<b>Select</b>	<b>Key</b>
Character left	SHIFT+LEFT
Character right	SHIFT+RIGHT
Word left	SHIFT+CTRL+LEFT
Word right	SHIFT+CTRL+RIGHT
Current line	SHIFT+DOWN if cursor is in column 1
Line above	SHIFT+UP if cursor is in column 1
Move cursor to column 1	CTRL+Q S (WordStar keystroke)
To end of line	SHIFT+END
To beginning of line	SHIFT+HOME
Screen up	SHIFT+PGUP
Screen down	SHIFT+PGDN
To beginning of file	SHIFT+CTRL+HOME
To end of file	SHIFT+CTRL+END



### Shortcut Keys to Find and Replace Text

Use either the Microsoft or WordStar keystrokes.

Search	Microsoft	WordStar
Find command	SHIFT+F3	CTRL+Q F
Find selected text	CTRL+F3	
Repeat last find	F3	
Find and replace		CTRL+Q A
Find matching brace, bracket, or parenthesis	CTRL+]	
View next tag	F2	
View previous tag	SHIFT+F2	







## The Context Operator

The form of the context operator is:

```
{[[<number>] <proc>] [, [<module>] [, [<exe>]]]}
```

Where:

<number> is the instance of the procedure on the stack (base 10).

n > 0 means count from the top of the stack down.

n < 0 means count from the current stack pointer up.

n = 0 means take first instance up (will find current procedure).

If no number is specified then the default is 0.

<proc> is the name of the procedure on the stack. If the name is overloaded, use argument types to distinguish the procedure.

<module> is the name of a module in the .exe to do the search in.

<exe> is the .exe or .dll name to search in.

You can use parentheses in any of the above constructs to include commas, etc.

There is also an abbreviated form of the context operator:

*module!*

Use the Debug command from the Options menu and check the Abbreviated Context debugger option to cause WinDbg to display contexts in this form. Even if this option is turned on, you can still enter contexts in the long form.

EXAMPLE:

```
// test.c
unsigned long ulVar = 100;
void main(void)
{
    unsigned char psz = "Hello World";
    printf(psz);
}
```

Assume the code above is compiled into hello.exe and the debugger has stepped to the printf() call in main(). Here are some valid expressions:

Expression	Meaning
? {*}psz	Display value of psz in the current (default) context.
? {main,test.c,hello.exe}psz	Display value of psz in procedure main from module test.c in image test.exe.
? {main,,}psz	Display value of psz in procedure main from current module in current image.
? {main,test.c,}psz	Display value of psz in procedure main from module test.c.



## Running WinDbg from the Command Line

When you run WinDbg from the command line, you can use the following options:

```
windbg [-a] [-g] [-h] [-i] [-k [platform port speed]] [-l[text]]  
[-m] [-p id [-e event]] [-s[pipe]] [-v] [-w name] [-y path]  
[-z crashfile] [filename.ext] [arguments]
```

Syntax	Explanation
-a	Ignore all bad symbols (but still print warning message).
-g	Go now; start executing the process.
-h	Causes child processes to inherit access to WinDbg's handles.
-i	Ignore workspace; like running without any registry data.
-k [ <i>platform port speed</i> ]	Run as a kernel debugger with the specified options: <i>platform</i> is the target machine type (i386, mips, alpha) <i>port</i> is the com port (com1 ... comn) <i>speed</i> is the com port speed (9600, 19200, 57600, ...)
-l [ <i>text</i> ]	Sets the window title for WinDbg.
-m	Start WinDbg minimized.
-p <i>id</i>	Attach to the process with the given id.
-e <i>event</i>	Signal an event after process is attached. Used only for post-mortem debugging (AeDebug).
-s [ <i>pipe</i> ]	Start a remote.exe server, using the named pipe.
-v	Verbose option; WinDbg prints module load and unload messages.
-w <i>name</i>	Load the named workspace.
-y <i>path</i>	Search for symbols along the specified path. You can specify multiple paths by separating them with semicolons.
-z <i>crashfile</i>	Debug the specified crash dump file.
<i>filename.ext</i>	Program to debug or file to edit.
<i>arguments</i>	Arguments to program being debugged.



### **Building a Program for WinDbg**

When you compile your program for WinDbg, use the following command-line options:

<b>Option</b>	<b>Effect</b>
/Zi	Causes the compiler to generate CodeView symbolic debugging information.
/Odi	Turns off optimizations. This will ensure that there is an exact correspondence between your source files and the compiler's output. If the compiler performs optimizations, it may rearrange or remove constructs, which will make it difficult or impossible to debug your program at the source level.

When you link your program, use the following command-line options:

<b>Option</b>	<b>Effect</b>
/debug:full	Adds public symbols, line numbers, program name, local variables, and types to the .exe file.
/debugtype:cv	Causes the linker to create CodeView-style symbolic debugging information.

Use the Open command from the Program menu to start debugging a newly-built executable.



### **Postmortem Debugging with WinDbg**

When a program crashes (for example, after a general-protection fault), Windows NT automatically invokes a debugger that is specified in the registry. To make WinDbg be that debugger, use regedt32.exe to edit the following registry key:

```
HKEY_LOCAL_MACHINE
  \SOFTWARE
    \Microsoft
      \Windows NT
        \CurrentVersion
          \AeDebug
```

Add or edit the Debugger value entry (REG\_SZ). Give it the following string value:

```
windbg -p %ld -e %ld
```

If you want the debugger to be invoked without a popup, add or edit the Auto value entry (REG\_SZ). Give it the string value 1. If Auto has the value 0, a popup will be displayed before the debugger is invoked.

You can also use the [Attach command](#) from the Run menu to attach the debugger to a process that is already running.



## Using Workspaces

After you have customized a debugging session for a program by opening and arranging various windows, setting breakpoints, and specifying options, you can save that information for future debugging sessions. The customized information is called a *workspace*, and you use the Save or Save As command from the Program menu to save the workspace for the program that you are debugging.

You can save several workspaces for the same program. For example, you might want to save one workspace for debugging a program's database module and a different workspace for debugging its UI module. WinDbg uses the default workspace for a program, unless you select a different one with the Open command from the Program menu. To make a particular workspace be the default for a program, check the Make Default box when you use the Save As command from the Program menu.

There is a common workspace that is not associated with a particular program. Before you open a program to be debugged, you can open various windows, size and arrange them, and specify options. Use the Save Common command from the Program menu to save that information as the common workspace.

Since the common workspace is not associated with a particular program, it does not include any program-specific information, such as breakpoints or watch expressions. You can use the common workspace as the starting point for the custom workspaces that you create for specific programs. When you start WinDbg, or open a program for the first time, WinDbg uses the common workspace to define its appearance and basic options.

If you want to use the current arrangement of windows and options to debug another program, you can select the current workspace when you use the Open command from the Program menu to open the new program. Like the common workspace, the current workspace does not include any program-specific information, such as breakpoints or watch expressions.

Remember to use the Save or Save As command from the Program menu to save any changes to a workspace that you want to keep. If you want to be prompted to save changes made to workspaces, check the appropriate box in the Environment command from the Options menu.

WinDbg stores the workspaces in the registry under the following key:

HKEY\_CURRENT\_USER

    \Software  
        \Microsoft  
            \WINDBG

Since the WinDbg configuration is stored on a per-user basis, each user of a machine can create a different set of workspaces for a particular program.

To restore WinDbg to its original configuration, delete the WINDBG key (use regedt32.exe) and restart WinDbg. This will delete all of the workspaces that the current user has set up.

If you want to start up WinDbg in its original configuration without deleting all of the workspaces stored in the registry, run it from the command line with the -i option:

```
windbg -i
```

If you want to start up WinDbg with a particular workspace, run it from the command line with the -w option:

```
windbg -w name
```

See also:

[Opening a Program](#)

[Saving a Program](#)

[Saving a Program under a New Name](#)

[Deleting a Program](#)

## Saving the Common Workspace



## **Opening a Program**

Use the Open command from the Program menu to open a new program or choose a new workspace for the current program. There can be only one program open at a time, so opening a new one will terminate and close the program that is currently open.

### **Programs**

This list box displays the fully-specified paths to all of the programs for which you have previously saved a workspace. Select the program that you want to debug, or select New to add a program to the list.

### **Workspaces**

This list box displays all of the workspaces associated with the currently selected program. Select the workspace that you want to open.

### **Common Workspace**

This is a workspace shared among all WinDbg programs. It does not contain program-specific information such as breakpoints.

### **Current Workspace**

Selecting this choice causes WinDbg to use the currently active workspace.

### **New**

Select New to add a program to the Programs list. After you add a new program, it will have associated with it two workspaces: Common and Current. The new program will not remain in the Programs list unless you save a workspace for it.

See also:

[Using Workspaces](#)

[Closing a Program](#)



### **Closing a Program**

Use the Close command from the Program menu to close the currently active program.

See also:

[Using Workspaces](#)

[Opening a Program](#)





### **Saving a Program**

Use the Save command from the Program menu to save changes to the current workspace. If the current workspace is new, WinDbg will invoke the dialog for the Save As command. If you want to be prompted to save changes made to workspaces, check the appropriate box in the Environment command from the Options menu.

See also:

[Using Workspaces](#)

[Deleting a Program](#)



### **Saving a Program under a New name**

Use the Save As command from the Program menu to save a new workspace, or to save the current workspace under a new name. To make the workspace be the default for the program, check the Make Default box.

See also:

[Using Workspaces](#)

[Deleting a Program](#)



### **Deleting a Program**

Use the Delete command from the Program menu to delete a program's workspaces. Select the program and the workspaces that you want to delete, and then select Delete. After you have deleted all of the workspaces that you want to delete, select OK.

#### **Programs**

This list box displays the fully-specified paths to all of the programs for which you have previously saved a workspace. Select the program whose workspaces you want to delete. If you delete all of a program's workspaces, the program name will be removed from the list box.

#### **Workspaces**

This list box displays all of the workspaces that have been saved for the currently selected program. Select the workspaces that you want to delete.

See also:

[Using Workspaces](#)

[Saving a Program](#)



### **Saving the Common Workspace**

Use the Save Common command from the Program menu to save the current configuration of windows and options not associated with a particular program as the common workspace. You can load the common workspace when opening a program by selecting Common Workspace from the Workspaces list box in the Program Open dialog.

The common workspace is also loaded when you open a new program, or when none of a program's workspaces is marked as the default.

See also:

[Using Workspaces](#)

[Opening a Program](#)



## Using Debugging Information Windows

Use the Window menu to display the following information windows as you debug a program.

Window	Information
Watch	Monitor variables and expressions set with the <u>Watch Expression</u> command.
Locals	Values of local variables within the function currently being stepped through. See <u>Locals Window</u> .
Registers	Current contents of the CPU registers.
Disassembly	Assembly-language instructions being debugged.
Command	Allows you to enter debugger commands with the keyboard. See <u>Command Window</u> .
Floating-Point	Current contents of floating-point registers and stack.
Memory	Current contents of memory. Display and format controlled by <u>Memory Options</u> dialog box. See also: <u>Using the Memory Window</u> .
Calls	Displays the current call stack and lets you jump to any stack frame displayed.



## Using the Memory Window

A Memory window lets you view a region of memory. You can open several Memory windows to view different regions.

When you open a Memory window, you specify an Address Expression that defines the base address of the region that the window displays. You can also select the format in which you want the memory contents displayed, whether the contents should fill the entire window, and whether the Address Expression should be "live" (reevaluated at each step of the debugger). To change the display options for a Memory window, change focus to that window and either use the Memory command from the Options menu or click on the last icon in the toolbar.

You can edit the contents of memory through a Memory window. Use the mouse or arrow keys to position the cursor to the location that you want to change and overwrite the existing data with the new data. If the data is displayed in byte format, any potential ASCII characters are also displayed at the righthand side of the window. You can edit either the numeric or the ASCII displays. WinDbg validates the data that you enter; you cannot enter invalid data for a memory location. Security and memory protections may prevent you from viewing or editing memory outside of the debuggee's scope.

You can edit the contents of memory through a Memory window. Use the mouse or arrow keys to position the cursor to the location that you want to change and overwrite the existing data with the new data. If the data is displayed in byte format, any potential ASCII characters are also displayed at the righthand side of the window. You can edit either the numeric or the ASCII displays. WinDbg validates the data that you enter; you cannot enter invalid data for a memory location. Security and memory protections may prevent you from viewing or editing memory outside of the debuggee's scope.

See also:

[Memory Options](#)



### **Setting the Memory Window Options**

When you open a new Memory window, you are presented with a Memory Options dialog box that lets you specify various options. You use the same dialog box to change the options for a Memory window that is already open. To call up the Memory Options dialog box for an existing window, change focus to that window and either use the Memory command from the Options menu or click on the last icon in the toolbar.

The dialog box lets you enter the following:

**Address Expression.**

This is an expression for the base address of memory region displayed in the window. The Address Expression can include a memory address, a variable, or a function address.

**Display Format.**

Select the format in which you want the memory contents displayed. If you select the byte format, any potential ASCII characters are also displayed at the righthand side of the window.

**Fill Window.**

Check this option to display as much as possible on each line.

**Reevaluate Expression Always (Live).**

Check this option to evaluate and update the memory window on all debug events.



### **Setting the Environment Options**

Use the Environment Options dialog to:

- Set tab stops and specify whether to keep tab characters or convert them to spaces.
- Make scroll bars visible. The scroll bars in some windows, like the Calls window, will not disappear when this option is changed. You must close and reopen the window to view it without the scroll bars.
- Specify the size of the undo/redo buffer.
- Tell WinDbg to search for a program along the path specified by the PATH environment variable.
- Be asked to save changes to a workspace if you have modified it.





## Setting and Removing Breakpoints

Breakpoints are useful when you have a general idea of where a bug occurs in a program. WinDbg runs the program until it hits a breakpoint, then stops. The breakpoint line is highlighted in the source window, and you can then step to the next line of code or trace through a function until you find the problem.

To set a breakpoint at a specific location:

1. Move the caret to the source-code line where you want to set the breakpoint.

If a single logical source line spans multiple physical lines, the breakpoint will be set on the last physical line of the statement or call. You should put the caret on or near the statement to set a breakpoint for the entire logical source line. WinDbg will search for statements around the current caret position if it cannot set a breakpoint at that position.

2. Click on the stophand icon on the Toolbar or press F9. The line containing the breakpoint will change color. If the line that the caret is on does not contain code or is the continuation of a line of code, the first line below it that contains code will be highlighted.

If you have source files with the same names in different directories, WinDbg sometimes will try to resolve the breakpoint set by using the incorrect source file. If this happens, rename one of the source files (along with any references to it), and then recompile.

To remove a breakpoint:

1. Move the caret to the highlighted source line that contains the breakpoint.
2. Click on the stophand icon on the Toolbar or press F9.

The breakpoint is removed and the line returns to its normal color.

The Breakpoints command from the Debug menu lets you set various options for breakpoints. You can set breakpoints based on Windows messages, memory location, and the values in an expression. You can also set and remove breakpoints from the Command window.

See also:

[The Context Operator](#)

[Command Window Reference](#)



## Unresolved Breakpoints

The Unresolved Breakpoint dialog is displayed when WinDbg cannot resolve a breakpoint that you have set. The dialog displays the breakpoint number, its status (such as Enabled [Unresolved]), and the context of the breakpoint.

The dialog lets you choose from the following options:

Command	Action
Clear	Clears the breakpoint.
Defer	Defers resolution until the next module-load.
Breakpoints	Calls up the Breakpoints dialog to resolve this breakpoint.
Disable	Disables the breakpoint.
Quiet Defer	Defers breakpoint resolution until you resolve the breakpoint. At each future module-load event, WinDbg tries to resolve the breakpoint. WinDbg will notify you if it cannot resolve the breakpoint.

A breakpoint specification may be ambiguous. For example, consider a C++ program that contains the following:

```
{,systhld.cxx,}Thread::Thread(int, char *)  
{,thread.cxx,}Thread::Thread(char *, int, int)
```

If you set the following breakpoint you will get the unresolved breakpoint dialog, since WinDbg puts priority on types:

```
> bp {,thread.cxx,}Thread::Thread
```

Also, you will not get an error when you set the following breakpoint:

```
> bp {,systhld.cxx,}Thread::Thread(char *, int, int)
```

WinDbg does not look at the context operator and sets the breakpoint in thread.cxx, rather than in systhld.cxx.

See also:

[The Context Operator](#)

[Command Window Reference](#)



## Breakpoints Dialog Box

Use the Breakpoints command from the Debug menu to set breakpoints with various conditions (you can click the stophand icon on the toolbar or press F9 to set an unconditional breakpoint at the current caret location). The Breakpoints dialog box lets you set the following conditions and options for a breakpoint:

### Break

A drop-down list box of the types of breakpoints:

#### At Location

This is the default type of breakpoint. The program will halt unconditionally at the location specified in the Location box. The default location is the source line that the caret is on.

#### At Location if Expr is True

The program will halt at the location specified in the Location box, if the expression in the Expression box is true.

#### At Location if Memory has Changed

The program will halt at the location specified in the Location box, if the memory pointed to by the expression in the Expression box has changed.

#### If Expr is True

The program will halt at the line it is on when the expression in the Expression box is true.

#### If Memory has Changed

The program will halt at the line it is on when the memory pointed to by the expression in the Expression box has changed.

#### At Wnd Proc

The program will halt at the Windows procedure specified in the Wnd Proc box.

#### At Wnd Proc if Expr is True

The program will halt at the Windows procedure specified in the Wnd Proc box, if the expression in the Expression box is true.

#### At Wnd Proc if Memory has Changed

The program will halt at the Windows procedure specified in the Wnd Proc box, if the memory pointed to by the expression in the Expression box has changed.

#### At Wnd Proc if Message is Received

The program will halt at the Windows procedure specified in the Wnd Proc box, if the message that you specify in the Messages dialog is received.

### Location

Enter the code location of the breakpoint. The default location is the source line that the caret is on.

You can specify the breakpoint location in any of the formats shown in the following table:

Format and Example	Description
<i>@linenumber</i> <i>@34</i>	Sets a breakpoint at line 34 in the active source window.
<i>{, source,}@linenumber</i> <i>{,GENERIC.C,}@34</i>	Sets a breakpoint at line 34 of GENERIC.C.
<i>function</i> <i>About</i>	Sets a breakpoint at the beginning of the About function.

See also: [The Context Operator](#)

### Wnd Proc

Enter the Windows procedure for the breakpoint. Select the procedure from the list or enter its name.

### Expression

Enter an expression to evaluate. The expression is true when it is nonzero.

### Length

Enter the size in bytes of the memory location pointed to by the expression in the Expression box.

### Pass Count

Enter the number of times that the breakpoint conditions must be met before WinDbg activates the breakpoint. After WinDbg activates the breakpoint, it will stop every time the breakpoint is hit; the count is not reset. The same is true for the */Pcount* option to the **bp** command.

When a breakpoint is disabled and then reenabled, its pass count is reset to the initial value. This gives you a way to set a breakpoint that is hit every *count* times. Just specify in the Command box the commands that will disable and reenable the breakpoint each time it is hit. For example, the following commands will disable and reenable breakpoint 0:

```
d0;be0
```

To set the equivalent breakpoint from the Command window, enter the following command (for breakpoint 0 at line 100, with a pass count of 5):

```
bp0 @100 /P5 /C"bd0;be0"
```

### Left

Displays the number of passes left before the breakpoint is activated. This field is initialized with the value in Pass Count and is decremented to zero. You cannot modify it directly. The value is initialized to Pass Count when:

- The breakpoint is added via the Add button or a **bp** command.
- The breakpoint is enabled via the Enable button or a **be** command.
- The breakpoint is instantiated via a module-load.

### Process

Enter the process in which you want to set the breakpoint. By default, the field is blank, causing the breakpoint to be set in the current process, or in process 0 if the debuggee has not been started.

### Thread

Enter the thread in which you want to set the breakpoint. By default, the field is blank, causing the breakpoint to be set in all threads in the current process.

### Commands

Enter the commands that you want WinDbg to execute when it hits the breakpoint. The commands are ones you would enter in the Command window; they affect only the thread that caused the breakpoint. Separate multiple commands with semicolons.

### Breakpoints

The list of breakpoints shows the state of all breakpoints for the current debugging session. A breakpoint entry in the list has the following fields:

#### Number

Displays the breakpoint number.

#### Action

Indicates whether breakpoint will be added (a), modified (c), or deleted (d) when you click OK.

#### Status

Indicates whether breakpoint is enabled (E), disabled (D), virtual (V), or has an unknown address (U). A

virtual breakpoint is one whose address has been obtained, but the breakpoint has not been committed; the breakpoint has not been set in memory. WinDbg does not stop for breakpoints marked with U or D.

**Definition**

Gives the breakpoint definition as it would be entered in the Command window. See Command Window Syntax.

**Messages**

This button is enabled only when the Break condition is set to At WndProc if Message is Received. It opens the Messages dialog box, which lets you select a single message or class of messages that WinDbg is to look for. See [Messages](#).

**Add**

Adds the breakpoint to the list of breakpoints.

**Clear**

Removes the selected breakpoint from the list of breakpoints.

**Clear All**

Removes all breakpoints for the current program.

**Enable**

Enables the currently selected breakpoint. The breakpoint status will change to enabled (E) in the breakpoint list box, and the Left field will be initialized to the value of the Pass Count field.

**Disable**

Disables the currently selected breakpoint. The breakpoint status will change to disabled (D) in the breakpoint list box.

**Modify**

Modifies the selected breakpoint.

To modify a breakpoint, select it in the Breakpoints list box, edit the fields that you want to change, and click the Modify button.

See also: [Unresolved Breakpoints](#)



## Controlling Program Execution

Once a breakpoint is reached and the program stops, you can control execution with commands from the Run menu. The following list shows the commands and their actions.

<b>Command</b>	<b>Action</b>
<u>Restart</u>	Resets execution to the first line of the program. It reloads the program into memory and discards the current values of all variables.
<u>Stop Debugging</u>	Terminates the debugging session and returns to a normal editing session.
<u>Go</u>	Executes code from the current statement until a breakpoint is reached, a breakpoint expression becomes true, or the end of the program is reached.
<u>Continue to Cursor</u>	Executes the program as far as the line that currently has the caret. This is equivalent to setting a temporary breakpoint at the caret location.
<u>Trace Into</u>	Steps into a function when it is called and then allows you to step through all of the instructions in the function.
<u>Step Over</u>	Single-steps through instructions in the program. If this command is used when you reach a function call, the function is executed without stepping into and through its instructions.
<u>Halt</u>	Halts the program and returns control to WinDbg.
<u>Go Handled</u>	Continues the execution of a program that has stopped at an exception, marking the exception as having been handled.
<u>Go Unhandled</u>	Continues the execution of a program that has stopped at an exception, without marking the exception as having been handled.
<u>Set Thread</u>	Selects the thread to monitor with the debugger. This determines which thread receives debugger commands and is viewed by the display windows. Also lets you freeze and thaw individual threads.
<u>Set Process</u>	Selects the process to monitor with the debugger. This determines which process receives debugger commands and is viewed by the display windows. Also freezes and thaws individual processes.
<u>Source Mode</u>	Toggles between source and assembly modes. This determines whether Trace Into and Step Over step through source or assembly instructions.



## Messages

When you set a breakpoint with the Break condition At WndProc if Message is Received, you must select a single message or class of messages that WinDbg is to look for. The Messages button in the Breakpoints dialog box calls up the Messages dialog box so that you can select the message. The Messages dialog box lets you select the following groups of messages:

### Message Type

Message Type lets you specify whether WinDbg is to break when a single message or one of a class of messages is received.

### Single Message

If you specified that WinDbg is to break when a single message is received, select the message in this drop-down list box.

### Message Class

If you specified that WinDbg is to break when one of a class of messages is received, check the classes of messages that you want WinDbg to look for.

### OK

Click OK to return to the Breakpoints dialog box. When you Add the breakpoint, the message or class of messages that you selected appear in the Breakpoints list box with the /M option. The following table lists the class identifiers for the /M option:

Identifier	Message Class
M	Mouse
W	Window
N	Input
S	System
I	Initialization
C	Clipboard
Z	Nonclient



## **Locals Window**

The Locals window shows variables local to the function that you are currently stepping through or stopped in. The window is updated whenever a variable's value changes AND control is returned to the debugger. When a context switch occurs, the Locals window displays the variables for the new context.

Use the Locals command from the Window menu to display the Locals window.

The Locals window consists of three parts, or "panes":

1. The left pane contains a "+" button for an expandable variable, such as a structure or array. Click the button to expand the variable. When you expand a variable, the "+" button changes to a "-" button. Click on the "-" button to collapse the variable. As you expand a variable, placeholder lines appear in the middle pane to show the expansion level.
2. The middle pane contains the names of the local variables. You can add a format descriptor to a variable by putting the cursor on the variable name and calling up the Locals Window Options dialog. To call up the dialog, click on the Options icon in the toolbar or use the Locals command from the Options menu.
3. The right pane displays the current values for the variables. The values are updated whenever control is returned to the debugger.

See also:

[Using Format Overrides](#)

[Using Watch Expressions](#)





## Using Watch Expressions

The Watch window lets you view the current values of expressions. The window is updated whenever an expression's value changes AND control is returned to the debugger.

Use the Watch command from the Window menu to display the Watch window.

The Watch window consists of three parts, or "panes":

1. The left pane contains a "+" button for an expandable expression, such as a structure or array. Click the button to expand the expression. When you expand an expression, the "+" button changes to a "-" button. Click on the "-" button to collapse the expression. As you expand an expression, placeholder lines appear in the middle pane to show the expansion level.
2. The middle pane contains the expressions. You can add a format descriptor to an expression by putting the cursor on the expression and calling up the Watch Window Options dialog. To call up the dialog, click on the Options icon in the toolbar or use the Watch command from the Options menu.
3. The right pane displays the current values for the expressions. The values are updated whenever control is returned to the debugger.

There are several ways to add an expression to the window:

- Type the expression into the middle pane.
- Cut and paste the expression from another window to the middle pane.
- Use the Watch Expression command from the Debug menu. If there is no Watch window, one will be created the first time you use the Watch Expression command.
- Use the Quickwatch command in the Debug menu (or, click the eyeglasses icon in the toolbar). Quickwatch displays a temporary version of the Watch window. You can add an expression from the Quickwatch dialog box to the Watch window. If there is no Watch window, one will be created the first time you add an expression from the Quickwatch dialog to the Watch window.
- If the expression is in the source window, put the caret on it. Use the Watch Expression command or call up the Quickwatch dialog. The expression will appear in the expression box, and you can add it from there to the Watch window.

There are several ways to remove an expression from the Watch window:

- In the Watch window:
  1. Highlight the expression
  2. Press Del
  3. Press Enter
- Use the Watch Expression command:
  1. Click on the expression
  2. Click Delete
  3. Click OK
- If you just added the expression through the Quickwatch dialog, you can click on the Remove Last button in that dialog to remove the expression from the Watch window.

See also:

[Using Format Overrides](#)

[QuickWatch](#)



## Using Quickwatch

Quickwatch gives you a fast way to view an expression. Unlike a Watch Expression, which is stored in the Watch window, a Quickwatch expression appears only in the Quickwatch dialog box. Use Quickwatch in exploratory debugging to check the values of expressions on the fly.

To use Quickwatch:

1. Stop the debuggee by hitting a breakpoint or using the Halt command from the run menu.
2. In the source window, put the caret on the variable that you want to examine.
3. From the Debug menu, choose Quickwatch. You can also click the eyeglasses icon on the Toolbar or press SHIFT+F9.

The Quickwatch dialog box displays three parts, or "panes":

1. The left pane contains a "+" button for an expandable expression, such as a structure or array. Click the button to expand the expression. When you expand an expression, the "+" button changes to a "-" button. Click on the "-" button to collapse the expression. As you expand an expression, placeholder lines appear in the middle pane to show the expansion level.
2. The middle pane contains the expression.
3. The right pane displays the current value for the expression. The value is updated whenever control is returned to the debugger.

The Quickwatch dialog box contains the following buttons:

### Evaluate

Evaluates the expression in the Watch box. Use this command when you type an expression in the box. You can add a format descriptor to an expression in the Watch box; see Using Format Overrides.

### Add Watch

Adds the expression to the Watch window. If there is no Watch window, one will be created.

### Remove Last

Removes the last expression added to the Watch window from the current Quickwatch.

See also: [Using Watch Expressions](#)



### **Watch Window Options**

Use the Watch command from the Options menu to set Watch window options. You can also click on the last icon in the toolbar when the Watch window has focus.

You can set the following options:

#### **Format**

Enter a format character for the item in the Watch window that contains the caret. See [Using Format Overrides](#).

#### **Window Options**

These control the initial display of the first level of expandable items, such as structures or arrays. Select the option that you prefer.



### **Locals Window Options**

Use the Locals command from the Options menu to set Locals window options. You can also click on the last icon in the toolbar when the Locals window has focus.

You can set the following options:

#### **Format**

Enter a format character for the item in the Locals window that contains the caret. See [Using Format Overrides](#).

#### **Window Options**

These control the initial display of the first level of expandable items, such as structures or arrays. Select the option that you prefer.



## Using Format Overrides

You can use format characters in the Watch, Quickwatch, Locals, and Command windows to override the default format for displaying data. These overrides are added to the variable name or expression as a suffix preceded by a comma.

### *format*

C-style format characters. For example, to display an int, use ",i" as the format specifier. Do not use percent signs (%).

Formats include:

Format	Definition	Example
D, d, I, i	Signed decimal integer.	> ?0x100,i 256
u	Unsigned decimal integer.	> ?-1,u 4294967295
o	Octal integer:	> ?40000,o 0o00001000000
x, X	Hexadecimal integer.	> ?40000,x 0x00040000
f	Signed value in floating point format with six decimal places (takes implicit/explicit type FLOAT as an argument).	> ?(float)(3./2.),f 1.500000
e, E	Signed value in scientific notation with up to six decimal places (trailing zeros and decimal point are truncated).	> ?(float)(3./2.),e 1.500000e+000
g, G	Signed value in floating point decimal format OR scientific format; whichever is more compact.	> ?(float)(3./2.),g 1.5
c	Single character.	> ?41,c 'A'
s	Characters up to the first null character.	> ?szHello,s Hello (szHello="Hello")
l	Long-Word modifier.	> ?3./2.,lf 1.500000
L	Long-Long Word modifier.	> ?0xffffffff,Li -1
h	Half-Word modifier:	> ?0x12345,hi 9029

### Note:

The "l", "L", and "h" modifiers work differently with different base types.

If, for instance, you are coercing an expression to type LONG DOUBLE (80 bit), the "L" modifier will work correctly on that expression. The "L" modifier also works correctly on LONG-LONG integer expressions, as shown in the table above.

WinDbg does not do implicit casting when evaluating an expression. It operates on the actual bit quantity unless coerced through casting, as shown above.





### **Restarting a Program**

The Restart command from the Run menu (SHIFT+F5) reloads the debuggee into memory, discarding any current variable values. WinDbg loads the program, but does not execute it. Breakpoints and watch expressions still apply after Restart.

You can use Restart whenever execution has stopped:

- At a breakpoint
- While single-stepping
- When normal execution is complete

After Restart, use the Go, Trace Into, Step Over, or Continue to Cursor commands to resume execution. Trace Into or Step Over will step to WinMain() or main() if such a symbol exists.



### **Terminating a Program**

Use the Stop Debugging command from the Run menu to end a debugging session.





## **Executing a Program**

Use the Go command from the Run menu (F5) to start or continue a program. The program proceeds from the current statement until one of the following occurs:

- The program reaches a breakpoint
- An exception occurs in the program
- The program ends



### **Executing to the Cursor**

Use the Continue to Cursor command from the Run menu (F7) to execute the program up to the line where the caret is positioned.

If WinDbg stops before reaching the line containing the caret, a subsequent Go will not stop execution at the caret position. You must use Continue to Cursor or again to stop at the line containing the caret.



### **Tracing Execution Into Routines**

Use the Trace Into command from the Run menu (F8) to single-step through the program, entering each routine as it is called.

When execution reaches a routine call, Trace Into steps from the statement that calls the routine to the first executable statement in the routine. You can then debug the routine using commands such as the Continue to Cursor.



### **Stepping Over Routines**

Use the Step Over command from the Run menu (F10) to single-step through the program, executing routines without stepping through their code. This saves you time by skipping routines you have already debugged.

If a breakpoint is hit, Go will not continue stepping; you must repeat the Step Over command.



### **Halting Execution**

Use the Halt command from the Run menu to stop execution temporarily and return control to the debugger. Use the Go command from the Run menu to continue execution.

The program may not halt immediately. For example, if you are currently executing system code, or are in a wait operation, the program will halt only after control has returned to the program's code.

Halt works only if the currently executing code is client-side.



## Set Thread

The Set Thread command from the Run menu displays a dialog box that contains a list of all threads connected to the current debugging session. The list shows the number, ID, and status of the threads.

From this list, you can select which thread to control with the debugger. You can also use this dialog to freeze or thaw (unfreeze) threads. To perform similar operations on processes, use the Set Process command.

Options include:

### Select

Makes the highlighted thread the current thread. When you select a thread, WinDbg updates the various debugging windows to reflect the current thread.

### Freeze

Freeze the highlighted thread. Use the Thaw button to resume the thread's previous state.

The Freeze button is changeable; when you freeze a thread or highlight a frozen thread, the button is relabeled to Thaw.

### Thaw

Resumes the highlighted thread. The thread will return to the state it was in before it was frozen.

The Thaw button is changeable; when you thaw a thread or highlight a thawed thread, the button is relabeled to Freeze.

### Freeze All

Freezes all threads in the list.

### Thaw All

Thaws all threads in the list.

## List Fields:

### TN

WinDbg's logical values to identify the threads. The current thread is marked with an asterisk (\*).

### TID

The system ID's for the threads, expressed in hexadecimal.

### Status

A letter, T or F, that indicates whether the thread is Thawed or Frozen, followed by a letter in parentheses that indicates whether the threads will be Thawed or Frozen after you click on OK.

The following table lists the possible thread states.

State	Explanation
Running	The thread is running.
Stopped	The thread is stopped.
Not Running	The thread has never been active or is at a breakpoint.
Except 1	The thread is stopped on a 1st-chance exception.
Except 2	The thread is stopped on a 2nd-chance exception.
Dead	The thread has exited, but the thread object has not yet been destroyed.
RIP Exception	The thread is stopped on a RIP. An RIP is generally issued from the Windows subsystem as a result of an API call that failed (for example, from a bad parameter or a failed validation).



### **Set Process**

The Set Process command from the Run menu displays a dialog box that contains a list of all processes connected to the current debugging session. The list shows the number, ID, and status of the processes. From this list, you can select which process to control with the debugger.

Options include:

Select

Makes the highlighted process the current process. When you select a process, WinDbg updates the various debugging windows to reflect the current process.

### **List Fields:**

PN

WinDbg's logical values to identify the processes. The current process is marked with an asterisk (\*).

PID

The system ID's for the processes, expressed in hexadecimal.

Status

The processes are either Running or Not Running.

See also: [Set Thread command](#)



## Attach

Use the Attach command from the Run menu to debug a running process. Highlight the process that you want to debug and click on Select.

If you are remotely debugging another machine, the Attach command lists the processes running on that machine.

You can also use the .ATTACH command from the Command window or the /P command-line option to attach WinDbg to a process.

To debug a user-mode driver running on another machine, select the CSR process, whose process ID is -1. When you debug the CSR process (which is the Windows subsystem), you must also do the following:

- Check the Disconnect on Exit option in the Debug command from the Options menu.
- If the target machine is running the free build of Windows NT, make sure the CSR-disable bit is not set on the target. Use regedt32 to edit the following registry key:

```
HKEY_LOCAL_MACHINE
  \SYSTEM
    \CurrentControlSet
      \Control
        \Session Manager
```

Edit the GlobalFlag value entry (REG\_DWORD) and clear the bit 0x0080000.

When a checked build of Windows NT is installed, the CSR-disable bit is 0, which lets you debug the Windows subsystem.





### **Go Handled**

Use the Go Handled command from the Run menu to mark an exception as having been handled and continue execution. The debuggee must have been stopped by an exception before you can use this command.

You can also use the [GH command](#) from the Command window.



### **Go Unhandled**

Use the Go Unhandled command from the Run menu to continue the execution of a debuggee that has stopped at an exception, without marking the exception as having been handled. The debuggee must have been stopped by an exception before you can use this command.

You can also use the GN command from the Command window.



### **Toggle Source/Asm Mode**

Use the Toggle Source/Asm Mode command from the Run menu to switch between source mode and assembly mode. In source mode, WinDbg steps line-by-line through the source code; in assembly mode, it steps instruction-by-instruction through the assembly code. The status bar displays whether the current mode is SRC or ASM and the toolbar icons are updated to reflect your choice.



## Starting a Remote Debugging Session

Remote debugging lets you debug a program from another machine. You run WinDbg on one machine (the *host*) and install the program you are debugging on another (the *target*). You run a remote monitor program (WinDbgRm) on the target machine to control the program you are debugging. WinDbgRm communicates with WinDbg across a serial cable or across the network.

This discussion assumes that you develop your program on the host and then move it to the target when you want to debug it.

To set up a remote debugging session, do the following:

1. If you are debugging across a serial cable, connect the cable to both the target and host.
2. On the target machine:
  - Install your program and its DLLs. You do not need to have the source files on the target.
  - Start WinDbgRm.
  - WinDbgRm starts minimized as an icon. Its default is to communicate with WinDbg across the network, using the named pipe "windbg". If you want to change how the host and target communicate, double-click on the WinDbgRm icon and select the Transport DLL command from the Options menu. Choose the appropriate transport layer in the Known Transport Layers list box.
3. On the host machine:
  - Start WinDbg from the Windows NT command prompt:  
start windbg *yourprog.exe*
  - Use the Debugger DLLs command from the Options menu to change the transport layer to match the one used by WinDbgRm on the target. If you are using named pipes, use the Change button to modify the transport layer parameters so that the first parameter is the name of the target machine.  
Select the Execution Model and Expression Evaluator to reflect the target machine.
  - Use the User DLLs command from the Options menu to specify the symbol search path to your program's symbols; use the Debug command from the Options menu to specify the source search path to your source files.
  - Select Go from the Run menu.

If you can start your program on the target before attaching the debugger to it, then you can change step 3 as follows:

3. On the host machine:
  - Start WinDbg
  - Use the Debugger DLLs command from the Options menu to change the transport layer to match the one used by WinDbgRm on the target. If you are using named pipes, use the Change button to modify the transport layer parameters so that the first parameter is the name of the target machine.  
Select the Execution Model and Expression Evaluator to reflect the target machine.
  - Use the User DLLs command from the Options menu to specify the symbol search path to your program's symbols; use the Debug command from the Options menu to specify the source search path to your source files.
  - Select the Attach command from the Run menu.
  - Select your program from the task list that the Attach command displays.

To connect to a machine with an IPC password, use the Windows NT NET USE command to manually connect to the remote IPC\$ share.

Use the .DISCONNECT command to disconnect the host from the target. Use the Attach command from the Run menu or the .ATTACH command to reestablish the connection. Another machine can run WinDbg and use .attach

to connect to the target and debug your program. If you are still connected and another machine tries to connect to the target, you will receive a popup notifying you of that fact. You have 20 seconds to cancel the popup, then you will be disconnected and the other machine will be connected as the host of the remote debugging session.

### **Win32S**

You can remotely debug a Win32S application. Everything works as described above, except that for WinDbgRm on the target you run wdbg32s.exe instead of windbgm.exe.

There are some limitations when you debug a Win32S application:

1. You cannot stop the debugger while the debuggee is running.
2. You cannot trace into 16-bit code. Do not try to circumvent this by setting a breakpoint in 16-bit code. Doing so will result in unpredictable behavior and might crash your machine.

If problems occur, check connections and make sure that WinDbg and WinDbgRm are using the same communications methods. If you are debugging across a serial cable and are having trouble maintaining the communications link between the host and target machines, reduce the transmission baud rate.

See also:

[Remote Debugging Cables](#)

[Choosing a Transport Layer](#)



### **Moving to a Specific Address**

The Goto Address command from the View menu opens a dialog box that asks for an address. This can be an expression such as a function, symbol, integer memory address, or any valid address expression. If the address is ambiguous, a list box with all ambiguous items is displayed. Unlike the BP command, only one address can be used.

After you select OK, WinDbg moves the cursor to the beginning of the function or address. If no Source window is open, WinDbg will open one.. If source-level debugging information is not available, WinDbg will open the Disassembly window and move the cursor to the address of the function or to the memory address.

To use the Goto Address command, do the following:

1. Compile the source file with debugging information.
2. Start debugging the program.
3. Select the Goto Address command from the View menu.
4. In the Address box, enter an address expression, such as &x.

Express addresses as you do in a program. For example, if you want to move to the location where "x" is stored, use the "&" operator. If you just type in "x", WinDbg will try to resolve an address from the contents of "x".

5. Choose OK.



### **Calls Window**

The Calls window displays the current call stack. You can double-click on a call and WinDbg will move you to the call location in the Source or Disassembly window.

The Calls window displays the most current call pushed onto the stack at the top of the listbox. You can use the Calls command from the Options menu to specify the information displayed in the Calls window.



### **Calls Window Options**

Use the Calls command from the Options menu to set Calls window options. You can also click on the last icon in the toolbar when the Calls window has focus.

You can select the following options to be displayed:

**Frame Pointer**

The pointer to the call frame on the stack.

**Return Address**

The address to which control is returned after the call.

**Function Name**

The name of the function in which the call was made.

**Displacement**

The displacement into the function at which the call was made.

**Parameters**

The parameters to the called procedure.

**First 4 DWORDS From Stack**

The first 4 DWORDS in the stack frame for the call.

**Source Information**

The source module and line number where the call occurred.

**Module Name**

The name of the executable. The format of this information is affected by the Abbreviated Context option that you set with the Debug command from the Options menu.

**Runtime Function Info**

Additional information for functions that have been compiled with FPO (frame pointer optimization: the /Oy option for the *cl386* compiler). If [EBP] appears in the additional FPO information, it means that the function uses the EBP register as a general-purpose register. This is likely if the function has nested "for" loops. The bracketed numbers in the FPO information indicate respectively: the number of DWORDs used for arguments, the number of DWORDs used for local variables, and the number of registers saved in the prolog.

**Maximum Frames**

The maximum number of frames that the Calls window will display.





## **Customizing the Environment**

Setting Debugging Options

Changing Debuggee Arguments

Changing Display Colors

Setting the Font, Font Style, Size

Removing the Toolbar, Status Bar, and Scroll Bars

Arranging Windows



### Remote Debugging Cables

If you are using software protocol, the serial cable that connects the host and target machines only needs to connect RXD to TXD, TXD to RXD, and GND to GND. WinDbg KD and remote debugging with WinDbg and WinDbgRm use this cable. The following table lists the pin numbers of these signals for 9-pin and 25-pin connectors.

Signal	9-pin Pin #	25-pin Pin #
RXD (Receive Data)	2	3
TXD (Transmit Data)	3	2
GND (Signal Ground)	5	7

If you are using hardware protocol, the serial cable that connects the two machines needs to connect RXD to TXD, TXD to RXD, DTR to DTR, GND to GND, and DSR to DSR. In addition, each side of the cable should connect DTR to DSR and RTS to CTS. The following table lists the pin numbers of these signals for 9-pin and 25-pin connectors.

Signal	9-pin Pin #	25-pin Pin #
RXD (Receive Data)	2	3
TXD (Transmit Data)	3	2
DTR (Data Terminal Ready)	4	20
GND (Signal Ground)	5	7
DSR (Data Set Ready)	6	6
RTS (Request to Send)	7	4
CTS (Clear to Send)	8	5



## Changing Display Colors

You can use color to improve the readability of different elements of a WinDbg session. For example, you can select different colors for different language elements in the source window, and different colors for the background and text of the various windows.

To change the colors of the Source window:

1. From the Options menu, choose the Color command.
2. Select Source Window in the Items list box. A checkmark will appear in front of any item that is selected.
3. Select the color that you want for the background of the source window and then choose Set Background. Select the color that you want for the text of the source window and then choose Set Foreground. The new color combination is shown in the Items list box.

To change the colors in the source window back to their default:

1. From the Options menu, choose the Color command.
2. Select Source Window in the Items list box.
3. Choose Set Default, then OK.

Notes:

1. You can change several items at the same time by selecting them in the Items list box. For example, you might want to set the background of several windows to the same color.
2. WinDbg uses only solid colors; it does not use dithered colors. If you select a dithered color, WinDbg will use the solid color displayed in the Color/Solid box in the dialog.
3. To revert all items back to their default values, choose Select All and then Set Defaults.



### **Setting the Font, Font Style, and Size**

You can use different fonts to give visual clues about the function of each of the windows in a WinDbg session. A given window can have only one font, font style (regular, bold, italic, bold italic), and size.

To change a font, font style, and font size:

1. Make the window whose text you want to change the active window.
2. From the Options menu, choose the Font command.
3. Select the font, style, and size from the appropriate list box.
4. Choose OK.

Notes:

1. If you check the Make Default option, then not only will the text in the active window be changed, but also each newly created window will use that font, style, and size.
2. Fixed-pitch fonts are the most useful. Proportional-spaced text can be hard to read in the context of programming.



### **Removing the Toolbar, Status Bar, and Scroll Bars**

You can increase the amount of the screen available to you for a WinDbg session by removing the toolbar, status bar, and scroll bars.

To remove the toolbar or status bar:

- From the View menu, choose the Toolbar or Status Bar command.

A checkmark appears next to these commands if the toolbar or status bar are visible.

To remove the scroll bars:

1. From the Options menu, choose the Environment command.
2. Uncheck Vertical and Horizontal in the Scroll Bars box.
3. Choose OK.

You can remove one or both scroll bars. If you remove the vertical scroll bar, use the up-arrow and down-arrow keys to scroll through a window.

The scroll bars in some windows, such as the Calls window, will not disappear when this option is changed. You must close and reopen the window to view it without the scroll bars.



## **Arranging Windows**

You can arrange your windows in an overlapped, side-by-side, or Codeview-style pattern.

To arrange the windows:

- For an overlapped pattern, choose Cascade from the Window menu.
- For a side-by-side pattern, choose Tile from the Window menu.
- For a Codeview-style pattern, choose Arrange from the Window menu.
- To arrange iconized windows, choose Arrange Icons from the Window menu.

If Overlay Source is checked, then any new source windows that you open with the New Window command will appear in the same location at which the first source window opened.

You can use standard Windows techniques to size and move windows.

To save an arrangement of windows, use the Save or Save As command from the Program menu to save the current workspace.



### **Changing the Debuggee Arguments**

You can change the command-line arguments passed to the debuggee when debugging starts.

To change the arguments:

1. From the Options menu, choose the Run command.
2. Enter the command-line arguments.
3. Choose OK.

WinDbg will ask if you want to perform a Restart. The changed arguments will not take effect until you perform a Restart.



## Debugger Options

You can control the behavior of WinDbg by choosing the Debug command from the Options menu. Set the options that you prefer and choose OK.

### Debugger options:

#### Ignore Case

Makes all references to symbols case-insensitive.

#### Display Segment

Shows selectors when addresses are displayed. This option does not affect the Disassembly window, which is controlled by the Disassembler section of this dialog.

#### Debug Child Processes

Causes WinDbg to debug processes started by the debuggee. Set this option before starting the debuggee.

#### Go on Thread Terminate

Do not stop the debugger when a thread exits.

#### Go on Process Attach

Do not stop the debugger when a new process is started.

#### Go on Child Create

Do not stop the debugger when a new thread is started.

#### Command Repeat

Lets you repeat the last command by pressing Enter.

#### Separate WOW VDM

Creates a separate Virtual DOS Machine (VDM) to run each 16-bit Windows (WOW) application. This prevents WinDbg from interfering with WOW applications that you don't want to debug. If you do not check this option, you might get the error "Cannot load *debuggee* because NTVDM is already running".

#### Alternate Single Stepping

Uses a different, slower method to single-step. Try checking this option if single-stepping doesn't work on an ill-behaved program.

#### Disconnect on Exit

Disconnects from the target machine when you exit WinDbg. Lets you quit WinDbg without crashing a remote debuggee.

#### Verbose Output

Displays detailed information about the modules that WinDbg loads.

#### Ignore Bad Symbols

Causes WinDbg to ignore bad symbol files. If this option is not checked, WinDbg displays a popup when it tries to load a bad symbol file. The popup will let you browse another directory for the correct symbols, add the new directory to the symbol search path, or ignore any further bad symbol files encountered.

#### Abbreviated Context

Causes WinDbg to display the debuggee name in the various windows as *debuggee*; for example GENERIC. Turning off this option causes WinDbg to display the full context of the debuggee name as {*source,executable*}; for example, {*generic.c,generic.exe*}. See [The Context Operator](#) for more details.

### Disassembler options:

#### Display Segment in Address

Show selectors when addresses are displayed.



#### Display Raw Bytes

Show the contents of memory corresponding to the disassembled instructions.

#### Uppercase Symbols and Opcodes

Display symbols and assembly-language instructions in uppercase.

#### Display Symbols

Show the assembly-language instructions with appropriate symbols. Turning off this option causes numeric constants and addresses to be shown in the disassembly listing.

#### Open Window on Demand

Prevents WinDbg from automatically opening a Disassembly window when it has no source file for a breakpoint.

### **Logfile options:**

#### Append

Causes the events and commands from the Command window to be appended to the named log file.

#### Open Automatically

Causes the named log file to be opened automatically, without your having to issue a .logopen command.

#### Name

Specifies the name of the logfile to which events and commands from the Command window are sent. The default is windbg.log.

### **Source Search Paths:**

Semicolon-separated list of additional directories to search for the source code of the debuggee.

### **Radix:**

Sets the default number base (radix) for expressions.

### **Registers:**

Selects which register sets are displayed in the Registers window.

### **Symbol Suffix:**

This option controls breakpoint-setting behavior. When WinDbg searches for a symbol (for example, CreateWindow), it will try appending "A" or "W" if it doesn't find the plain name.



## Debugger DLLs

WinDbg uses specific DLLs to determine

- how to handle symbols
- how to evaluate expressions
- how to communicate with a target machine when debugging remotely
- which execution model to use for the target machine when debugging remotely

To change the way WinDbg handles any of these items:

1. From the Options menu, choose the Debugger DLLs command.
2. Select the appropriate DLL for the Symbol Handler, Expression Evaluator, Transport Layer, and Execution Model.
3. Choose OK.

The list boxes display a short name and a description for each available DLL. To see the actual filename and any parameters passed to the DLL, highlight the description of the DLL in the list box and choose Change. The Change dialog displays and lets you change the short name, description, filename, and parameters for the DLL.

To add a new DLL to a list of choices, choose Add and fill in the appropriate information.

To delete a DLL from a list of choices, highlight the description of the DLL and choose Delete.

### Symbol Handler

WinDbg uses the Symbol Handler DLL to access symbolic debugging information from the program being debugged.

### Expression Evaluator

WinDbg uses the Expression Evaluator DLL to evaluate expressions, such as breakpoint expressions. If you are debugging remotely, select the DLL that corresponds to the target machine. For example, if you are debugging a MIPS target machine with C++ expressions, choose eecxxmip.dll (C++ with MIPS register set).

### Transport Layer

WinDbg uses the Transport Layer DLL to control communication between the host and target machines during a debugging session. Choose the DLL that corresponds to the one used by WinDbgRm on the target machine. There are several transport layer DLLs to choose from:

- tlloc.dll

Choose tlloc.dll (Debugging on same machine) if you are debugging a program on the same machine that you are running WinDbg on.

Parameters: none

- tlpipe.dll

Choose tlpipe.dll (Named pipe: host=targethost, pipe=windbg) if you are remotely debugging a machine over a network connection. Use this DLL on both the host and the target machines.

Parameters:

*Target Machine Host Name* (default is targethost)<space>*Pipename* (default is windbg).

- tlser.dll

Choose `tlser.dll` (Serial - COM1, *baudrate*) if you are remotely debugging a machine over a serial debugging cable. Use `tlser.dll` on the WIN32 side of the remote debugging session; use `tlser32s.dll` on the WIN32S side.

Parameters: `COMx:baudrate`

- `tlser32.dll`

Choose `tlser32.dll` (Serial - COM1, *baudrate*) if you are remotely debugging a Win32S application. machine over a remote debugging cable. Use `tlser32.dll` on the Win32 side of the remote debugging session; use `tlser32s.dll` on the Win32S side.

Parameters: `COMx:baudrate<space>[XON]`

If you do not specify XON, the default protocol is hardware. The XON/XOFF protocol lets you use a three-wire serial cable; the hardware protocol requires a more complicated cable, but permits faster communication between the host and target machines.

## Execution Model

WinDbg uses the Execution Model DLL to specify the native execution model of the target machine. For example, if you are debugging a MIPS target machine, choose `emmip.dll` (MIPS CPU).



## User DLLs

You can control the way WinDbg loads symbols for DLLs that are called by the debuggee.

Once you have run the debuggee, choose the User DLLs command from the Options menu. The dialog lists the DLLs called by the debuggee and shows their current status. The fields in the list are as follows:

### Module Name

Lists the filename of each of the DLLs called by the debuggee. If you select a filename, the full path to that DLL appears in the box immediately below the list box.

### Load Plan

The Load Plan for a DLL is a combination of the Location and Symbols selections at the bottom of the dialog, which are described below.

### Current Status

The Current Status for a DLL indicates whether the DLL is loaded and the status of its symbols, if they exist.

Status	Meaning
Not loaded	The DLL has not been loaded.
Symbols loaded	The DLL has been loaded, along with its symbols.
No symbols loaded	The DLL has been loaded, but not its symbols.
Symbol loading deferred	The symbols for the DLL will be loaded when they are needed.
Symbol loading suppressed	The symbols for the DLL will not be loaded.
Symbol converted & loaded	WinDbg found COFF symbols, converted them to Codeview, and loaded them.

To change the set of directories that WinDbg searches for symbols, add or delete the directory names that you want in the Symbol Search Path. Separate the different directory names with semicolons.

To add to the list of DLLs loaded by the debuggee:

1. Enter the name of the new DLL (or select it via the Browse button) in the New DLL name box.
2. Select one of the following in the Location box:
  - Local (the DLL is on your host machine)
  - Remote (the DLL is on the target machine)
3. Select one of the following in the Symbols box:
  - Load (load the symbols for the DLL when the DLL is loaded)
  - Defer (load the symbols for the DLL only when they are needed)
  - Suppress (do not load the symbols for the DLL)
4. Choose Add.

To delete a DLL from the list, highlight the DLL and choose Delete.

To change the Load Plan for a DLL:

1. Highlight the DLL or DLLs in the list box.
2. Make the appropriate selections in the Location and Symbols boxes.
3. Choose Modify.

You can change the default Load Plan for new DLLs by making the appropriate selections in the

Location and Symbols boxes and then choosing Default.

To close the dialog and apply any changes that you have made, choose OK.

To close the dialog and cancel any changes that you have made, choose Cancel.



## Exceptions

The Exceptions command from the Options menu lets you determine WinDbg's behavior when a user-defined or system exception occurs.

The Exceptions dialog contains the following parts:

### Except #, Name

The Except # and Name boxes show the exception number and name of an exception that you have selected from the Exception List (see below). They also let you enter the number and name of a user-defined exception that you are adding to the list.

### Action

The Action list box lets you select one of the following actions for WinDbg to take for the exception you have selected or are adding:

- Enabled (WinDbg breaks before the exception is passed to the debuggee)
- Notify (WinDbg displays a message in the Command window and passes the exception to the debuggee)
- Disabled (WinDbg does not break; it passes the exception to the debuggee)

### First Chance Command

If the exception is Enabled, WinDbg breaks and runs any commands that you specified for the exception in the First Chance Command box.

### Second Chance Command

If neither WinDbg nor the debuggee has handled an exception, then WinDbg will break and run any commands that you specified for the exception in the Second Chance Command box.

### Exception List

The Exception List shows the Exception Number and the Name of the exceptions that can trigger WinDbg. Several system exceptions appear in the list by default.

The following table shows what you need to specify in the Exceptions dialog in order for WinDbg to take a given action for an exception.

WinDbg Behavior on Exception	You Specify
Break.	Action = Enabled
Break and run a list of commands.	Action = Enabled First Chance Command = list of commands
Do not handle the exception; pass it to the debuggee.	Action = Disabled
Do not handle the exception; display a message and pass the exception to the debuggee.	Action = Notify

To add an exception to the Exception List:

1. Enter an exception number and name in the Except # and Name boxes. If an exception with that number already exists in the list, the new name will replace the existing name.
2. Select the appropriate Action for the exception.
3. Specify any First Chance Commands and Second Chance Commands that you want WinDbg to run.

4. Choose Add.

To delete an exception from the list:

1. Select the exception that you want to delete.
2. Choose Delete.

To modify an exception:

1. Select the exception that you want to modify.
2. Change the exception number, name, action, first chance command, or second chance command, as appropriate.
3. Choose Modify.



### **Source Not Found**

This dialog box appears when WinDbg is unable to find a file in one of the current paths. Choose Ignore to have WinDbg continue without finding the file. Choose Browse for Source File to have WinDbg open a dialog box that lets you select the file to load.

You can specify a default source search path in the Debugger Options dialog box.





### **Browse for Source File**

This dialog box lets you select the file to load.

You can specify a default source search path in the Debugger Options dialog box.



## **Kernel Debugging**

[Setting up a Kernel Debugging Session](#)

[Serial Cable](#)

[Free and Checked Builds of Windows NT](#)

[Enabling Kernel Debugging on the Target](#)

[Accessing the Target's Symbols](#)

[Starting WinDbg KD](#)

[Kernel Debugger Options](#)

[Other Useful Debugger Options](#)

[Connecting to a Running Target](#)

[Debugging a Crash Dump](#)

[Kernel Debugging Extensions](#)

[Creating Extensions](#)

[Building Your Driver](#)



### Setting up a Kernel Debugging Session

You can debug kernel-mode drivers with WinDbg. Install your driver on one machine (the *target*) and run WinDbg KD (WinDbg with the kernel debugging option enabled) on another machine (the *host*). The host and target communicate with each other across a serial cable.

To set up a kernel debugging session, do the following:

1. Connect the host and target machines with a null-modem serial cable. Plug the cable into COM1 on the host machine and an unused serial port on the target machine.
2. On the target machine:
  - Install the checked build of Windows NT. See Free and Checked Builds of Windows NT.
  - Install your driver.
  - Enable kernel debugging. See Enabling Kernel Debugging on the Target.
3. On the host machine:
  - Install the free build of Windows NT.
  - Install the SDK (which contains WinDbg).
  - Copy ntoskrnl.dbg, hal.dbg, and your driver from the target machine to the host machine.
  - Copy your driver's source to the host machine.
  - Start WinDbg KD. See Starting WinDbg KD.
  - Use the User DLLs command from the Options menu to specify the symbol search path to where you copied ntoskrnl.dbg, hal.dbg, and your driver.
  - Use the Debug command from the Options menu to specify the source search path to where you copied the source for your driver.
  - Select Go from the Run menu and wait for KD: waiting to connect... to appear in the Command window.
4. Boot the target machine and select the version of Windows NT for which you enabled kernel debugging.
5. To break into WinDbg KD, press CTRL+C on the host machine, or press SYSRQ on the target machine. To continue execution, select Go from the Run menu.



### Serial Cable

For kernel debugging, the serial cable that connects the host and target machines only needs to connect RXD to TXD, TXD to RXD, and GND to GND. The following table lists the pin numbers of these signals for 9-pin and 25-pin connectors.

Signal	9-pin Pin #	25-pin Pin #
RXD (Receive Data)	2	3
TXD (Transmit Data)	3	2
GND (Signal Ground)	5	7



## Free and Checked Builds of Windows NT

The DDK provides two versions of the Windows NT operating system, called the *free build* and the *checked build*. The DDK *Getting Started* manual explains how to install each version.

The free build of Windows NT is the retail version that end users receive. Free binaries are built with full optimization and contain minimal debugging symbols, such as function entry points and global variables.

The checked build is shipped only with the DDK; it is used in debugging drivers and other system code. Checked binaries provide error checking, argument verification, and system debugging code not present in the free binaries.

Much of the additional code in the checked binaries is in the form of ASSERT macros that test an expression. If the expression evaluates to FALSE, the macro generates a kernel debugger error message and breaks into the debugger. This lets you immediately determine the cause and location of the error. For example, recursive acquisition of a spin lock (where a thread tries to acquire a spin lock that it already owns) hangs the system in the free build, but generates a breakpoint on the checked build.

Use the free and checked builds of Windows NT as follows:

- Use the free build on the host machine, where you run WinDbg.
- Use the free build when you develop and build your driver. Performance suffers in the checked build due to the additional code that is executed.
- Use the checked build on the target machine when you test and debug your driver during the initial phases of its development.
- Use the free build on the target machine when you test and debug your driver during the final phases of its development. You *must* do the final testing of your driver with a free version of the driver on the free build of Windows NT.

The extra protection of the checked build of Windows NT carries some penalties. Because of the additional code, the checked binaries are larger and run slower than the free binaries. This can conceal synchronization problems, such as race conditions, that become apparent only when you use the free build.

If you run the checked build of Windows NT without having enabled kernel debugging (see [Enabling Kernel Debugging on the Target](#)), unexpected system shutdowns can occur. This is because the additional checks in the checked build increase the likelihood of encountering a breakpoint. A kernel-mode breakpoint with kernel debugging disabled is an unhandled kernel-mode exception, so the system calls the KeBugCheck function and shuts down.

If kernel debugging is enabled, however, a breakpoint freezes all threads in the target system except for the component that communicates with WinDbg KD. This lets WinDbg KD connect to a running target machine that has kernel debugging enabled, even after a breakpoint has been encountered. See [Connecting to a Running Target](#) for details.



## Enabling Kernel Debugging on the Target

### Note:

Do not confuse this procedure with the one that starts WinDbg as the kernel debugger, WinDbg KD. See [Starting WinDbg KD](#) for details.

You must enable kernel debugging on the target machine before you can connect to it with WinDbg KD. The procedure for enabling kernel debugging on the target machine is different for different types of machines.

For an ARC machine (MIPS, ALPHA, etc.), you modify the `OSLOADOPTIONS` environment variable in the ARC firmware; for an x86-based machine, you modify a line in the [operating systems] section of `boot.ini`. For either machine, you can specify the following options:

#### DEBUG

Enables kernel debugging.

#### NODEBUG

Disables kernel debugging. This is the default.

If you specify NODEBUG, then DEBUGPORT, BAUDRATE, and CRASHDEBUG are ignored.

#### DEBUGPORT=*Port*

Specifies the serial port (*Port* is COM1, COM2, etc.) used on the target machine.

If you specify DEBUGPORT, kernel debugging is enabled; you do not also have to specify DEBUG.

#### BAUDRATE=*BaudRate*

Specifies the baud rate (*BaudRate* is 9600, 19200, etc.) used by the target machine. Use the highest rate that works for your machines.

If you specify BAUDRATE, kernel debugging is enabled; you do not also have to specify DEBUG.

#### CRASHDEBUG

Causes the debugger to activate only when the system bugchecks.

#### MAXMEM=*SizeInMB*

Specifies the amount of memory to be made available to the system. For example, you can restrict a 16 MB machine to 10 MB.

#### SOS

Causes the OS loader to display the names of the drivers as they load when Windows NT is booting.

### Examples

For an x86 machine, the following line in `BOOT.INI` would boot Windows NT in a simulated 10 MB environment, with kernel debugging enabled on COM1, and communicating at 115200 baud:

```
multi(0)disk(0)rdisk(0)partition(1)\NT="Windows NT" /MAXMEM=10 /DEBUGPORT=COM1  
/BAUDRATE=115200
```

For an ARC machine, the following `OSLOADOPTIONS` would accomplish the same thing:

```
OSLOADOPTIONS= MAXMEM=10 DEBUGPORT=COM1 BAUDRATE=115200
```



### Accessing the Target's Symbols

WinDbg KD must have access to the symbols for the target machine. The symbols for the Windows NT kernel and drivers are stored in .dbg files; the symbols for your driver are stored in the driver's .sys file itself, or may have been split off into a .dbg file. Copy the following files from the target machine to the host machine, or to another machine on the network to which the host has access.

- ntoskrnl.dbg
- hal.dbg
- *yourdriver.sys* (or *yourdriver.dbg*, if the symbols have been split off into a separate .dbg file)
- *otherdrivers.sys* with which your driver interacts (or *otherdrivers.dbg*, if the symbols have been split off into separate .dbg files)

Specify the symbol search path with the User DLLs command from the Options menu of WinDbg. If you start WinDbg KD from the command line, you can use the [-y command-line option](#) to specify the symbol search path.



## Starting WinDbg KD

There are several methods that you can use to start WinDbg as the kernel debugger, WinDbg KD. They all accomplish the same thing.

In the first method, start WinDbg KD from the command prompt:

```
start windbg -k platform port speed
```

where *platform* is the target machine type (i386, mips, alpha), *port* is the com port (com1 ... comn) to which the serial cable is attached, and *speed* is the com port speed (9600, 19200, 57600, ...). See [Running WinDbg From the Command Line](#) for a description of all of the command-line options.

In the second method, start WinDbg from the command prompt with ntoskrnl or ntoskrnl.exe as a command-line option:

```
start windbg ntoskrnl[.exe]
```

Use the [Kernel Debugger command](#) from the Options menu to specify the target platform, com port, and speed.

In the third method, start WinDbg by double-clicking on the WinDbg icon in the Program Manager. Use the Kernel Debugger command from the Options menu to enable kernel debugging and specify the target platform, com port, and speed.

Once you have set up a kernel debugging session with the appropriate parameters, save it as a named workspace with the Save As command from the Program menu. You can then start a kernel debugging session with those parameters by specifying the workspace name on the command line:

```
start windbg -w workspacename ntoskrnl
```





## Kernel Debugger Options

Use the Kernel Debugger command from the Options menu to specify options for kernel debugging. You can specify the following:

### Stop at Initial Breakpoint

Check this option to set an initial breakpoint in the kernel for WinDbg KD to stop at.

### Enable Kernel Debugging

Check this option to enable kernel debugging.

### Go on Exit

Check this option to cause WinDbg KD to issue a Go command to the target when you exit WinDbg KD.

### Baud Rate

Select the baud rate for communicating with the target. This should be the same as the rate that you specify when you enable kernel debugging on the target. See [Enabling Kernel Debugging on the Target](#) for details. Use the highest rate that works for your machines.

### Port

Select the com port to which the serial cable is attached.

### Cache Size

Select the size of the cache for WinDbg KD to use at startup for memory values. Normal operations, such as single-stepping or the G command, invalidate the cache. If WinDbg KD has frozen the target machine, but hardware on the target can change memory (for example, through shared memory or DMA), you must disable the cache to see the memory changes. Use the [.cache command](#) to examine the current cache settings, specify a new cache size (in KB), or disable the cache (.cache 0). The default cache size is 100 KB.

### Platform

Select the target machine type (x86, MIPS, ALPHA).

### Crash Dump

Enter the name of the crash dump file (the default is memory.dmp) or click on the Browse button to select a crash dump file. See [Debugging a Crash Dump](#) for information on debugging crash dumps.



### **Other Useful Debugger Options**

You can set several options with the Debug command from the Options menu that are useful with WinDbg KD.

Make sure that Alternate Single Stepping is not checked, as this option causes WinDbg to use a single-stepping algorithm that slows down kernel debugging.

Turn off Ignore Bad Symbols if you want to receive a popup when WinDbg tries to load a bad symbol file. The popup will let you browse another directory for the correct symbols, add the new directory to the symbol search path, or ignore any further bad symbol files encountered.

Check the Disassembler option Open Window on Demand. If this option is not checked, and there is no source file for a breakpoint, WinDbg KD will automatically open a Disassembly window and read in a block of machine code from the target to display as the source for the breakpoint. This is a time-consuming operation that slows down kernel debugging.

To generate a log of events from the Command window, specify a logfile. You can append to or overwrite an existing logfile.



### **Connecting to a Running Target**

You can connect WinDbg KD to a target machine that has already booted. This is especially useful when an error or exception causes the target to crash. To connect to a running target, do the following:

Start WinDbg KD on the host. See [Starting WinDbg KD](#).

WinDbg KD either connects immediately or waits to connect, depending upon what has happened on the target. If WinDbg KD is waiting to connect, press CTRL+C on the host or SYSRQ on the target.

Reload the symbols for the target by running [!reload](#) in the Command window of WinDbg KD on the host.



## Debugging a Crash Dump

You can debug a crash dump with WinDbg KD. Specify the name of the crash dump file in the Kernel Debugger Options dialog box. If you start WinDbg KD from the command line, you can use the -z command-line option to specify the name of the crash dump file.

To configure the target to generate a crash dump, use the System applet in the Control Panel. Click on the Recovery button and then click on the option that enables writing debugging information to a file when a STOP occurs. That file is the crash dump file. Its default name is memory.dmp.

There are two utilities shipped with the DDK that are useful in debugging crash dumps:

**DUMPCCHK** *[[options]] CrashDumpFile*

### Parameters

*options*

- ? Displays a help message.
- v Verbose mode.
- p Print header only, do not validate file.
- q Perform a quick test.

*CrashDumpFile*

The name of the crash dump file.

### Description

This utility checks the validity of a crash dump file.

**DUMPREF** *[[options]] SourceDumpFile ShareName SymbolPath*

### Parameters

*options*

- ? Displays a help message.
- d Destination dump file name.

*SourceDumpFile*

The name of the crash dump file.

*ShareName*

The name of the network share where the crash dump file is located.

*SymbolPath*

The path to the symbols for the crash dump file.

### Description

This utility creates a small reference file that points to the actual crash dump file. You can use the reference file as though it were the crash dump file; WinDbg KD will find the actual crash dump file out on the network. This lets you send the reference file, which is only a few bytes long, to someone for debugging, rather than sending the actual crash dump file, which may be many megabytes long.



## Kernel Debugging Extensions

WinDbg KD has a number of built-in extension commands that are useful in debugging drivers. The command-line syntax for a built-in extension is:

**!extension\_name** [*arguments*]

The following table lists the built-in extensions. You can also enter **!?** at the WinDbg KD prompt for a list of available extensions. See also [Creating Extensions](#).

Syntax	Description
<b>!cxr</b> <i>address</i>	Displays the context record at the specified address.
<b>!db</b> [ <i>PhysicalAddress</i> ]	Displays a hexadecimal and ASCII dump of 128 bytes from the specified physical memory address on the target machine. If the <i>PhysicalAddress</i> argument is omitted, the command dumps beginning at the byte following the previous <b>!db</b> dump.
<b>!dd</b> [ <i>PhysicalAddress</i> ]	Displays a hexadecimal dump of 32 <b>ULONGs</b> from the specified physical memory address on the target machine. If the <i>PhysicalAddress</i> argument is omitted, the command dumps beginning at the byte following the previous <b>!dd</b> dump.
<b>!default</b> <i>DllName</i>	Change the default extension DLL.
<b>!devobj</b> <i>address</i>	Displays a list containing information about a specified device object. The <i>address</i> argument is a pointer to the device object, and can be obtained using the <b>!drvobj</b> command. The information displayed includes the device name of the object, information about the device's current IRP, and a list of addresses of any pending IRPs in the device's queue.
<b>!drvobj</b> <i>address</i>	Displays a list of all device objects created by a specified driver. The <i>address</i> argument is a pointer to the driver object, and can be obtained by examining the arguments passed to the driver's DriverEntry routine.
<b>!drivers</b>	Displays information about all loaded drivers.
<b>!ed</b> <i>PhysicalAddress</i> <i>ulong0</i> [ <i>ulong1</i> <i>ulong2</i> ...]	Writes one or more <b>ULONG</b> values to the specified physical memory address on the target machine.
<b>!errlog</b>	Displays information about any pending events in the I/O system's error log. These are events queued by calls to the <b>IoWriteErrorLogEntry</b> function to be written to the system's event log for subsequent viewing by the Event Viewer. This command only displays pending entries that have not been written to the event log. If the system crashes, this function is useful for viewing events that are not written to the event log because of the crash.
<b>!exr</b> <i>address</i>	Displays the formatted contents of the EXCEPTION_RECORD pointed to by <i>address</i> .
<b>!frag</b> [ <i>flags</i> ]	Displays information about fragmentation in the nonpaged pool. The two low-order bits of the <i>flags</i> argument specify the level of detail displayed.
<b>!filecache</b>	Displays information about the file system cache.
<b>!handle</b> [ <i>handle</i> [ <i>flags</i> [ <i>process</i> ]]]	Displays data about <i>handle</i> , the index of the handle in the process's handle table. If <i>handle</i> is 0 (or unspecified), the command dumps data for all handles. The two low-order bits of the <i>flags</i> argument specify the level of detail displayed. The <i>process</i> argument identifies the process to dump handles for, and is either the process ID or the address of a process object. If not specified, the current process is used. 0

	dumps information for all active processes.
<b>!heap</b> [ <i>address</i> [ <i>flags</i> [ <i>process</i> ]]]	Dumps the heap for a process.
<b>!irp</b> <i>address</i>	Displays formatted information about the IRP specified by <i>address</i> . Note that the function does not verify that <i>address</i> is an IRP. This command can dump IRPs after they have been freed or completed, or in a post-mortem situation.
<b>!irpzone</b> [F]	Displays information about all IRPs allocated from zoned pool. If the 'F' argument is omitted, the command prints only the address of the IRP, the address of the thread object associated with the IRP, and a pointer to the IRP's current device object (for example, the device associated with the last stack).
<b>!kb</b>	Displays a stack trace from the last frame dumped by !cxr or !trap.
<b>!load</b> <i>DllName</i>	Load the named extension DLL.
<b>!locks</b> [-v] [ <i>address</i> ]	Displays information about a kernel-mode resource lock specified by <i>address</i> . If <i>address</i> is omitted, the command dumps information on all kernel-mode resource locks. Use the -v option for verbose mode.
<b>!lpc</b>	Displays information about all LPC port objects.
<b>!memusage</b>	Displays summary statistics on physical memory usage, collected from the page frame database.
<b>!noversion</b>	Disable version checking for extension DLLs.
<b>!object</b> <i>address</i>	Displays information about the object pointed to by <i>address</i> .
<b>!pcr</b>	Displays the formatted contents of the PCR.
<b>!pfn</b> <i>PageFrameNumber</i>	Dumps the page frame database entry for the physical page identified by <i>PageFrameNumber</i> , which is the page's index in the array of records of the page frame database.
<b>!pool</b>	Dumps the kernel-mode heap.
<b>!process</b> [ <i>process</i> [ <i>flags</i> ]]	Dumps information about the process specified by <i>process</i> , which is either the process ID or the address of the process object. If not specified, the current process is used. 0 dumps information for all active processes. The three low-order bits of the <i>flags</i> argument determine the level of detail displayed.
<b>!pte</b> <i>VirtualAddr</i>	Displays information about the physical address that corresponds to the specified virtual address. This is useful if a driver does a lot of translations back and forth from virtual to physical. The PFN address is the page frame number of the virtual address. Use this as the first 20 bits of the physical address, and append the last 12 bits of the virtual address to come up with the physical address.
<b>!reload</b> [ <i>imagename</i> ]	Causes WinDbg KD to load the symbol files, discarding any previously loaded symbols. You can specify the name of an image file whose symbols you want WinDbg KD to reload.
<b>!sel</b> [ <i>selector</i> ]	Displays selector values.
<b>!srb</b> <i>address</i>	Displays the formatted contents of the SCSI_REQUEST_BLOCK pointed to by <i>address</i> .
<b>!sympath</b> <i>SymbolPath</i>	Resets the path used by WinDbg KD to find debugging symbols.
<b>!thread</b> [ <i>thread</i> [ <i>flags</i> ]]	Dumps information about a specified thread, where <i>address</i> points to a thread object. If the <i>thread</i> argument is omitted, the command dumps

data for the current thread. The three low-order bits of the *flags* argument determine the level of detail displayed.

**!time**

Displays the PerformanceCounterRate and TimerDifference.

**!trap** *TrapFrameAddr*

Dumps the machine state when the trap frame occurred. On x86-based platforms, a trap frame is generated whenever there is an interrupt or a system call. Identify a trap frame by the value of its first argument (badb0d00). Use the kv command to find a trap frame. This is useful if you want to see the state of the machine when an access fault occurred.

**!tss**

Displays the contents of the Task State Segment.

**!unload**

Unloads the default extension DLL.

**!vad**

Dumps VADs.

**!vm**

Displays summary statistics about virtual memory usage.



## Creating Extensions

You can create your own debugging commands by writing an extension DLL. You might want to write a command, for example, to dump a complex data structure.

To run a command called `foo`, which is an exported function from `mydll.dll`, enter the following at the WinDbg command prompt:

```
!mydll.foo [args]
```

WinDbg will load `mydll.dll`, call the entry point `foo`, and pass `args` to `foo`. Once WinDbg has loaded `mydll.dll`, you can run an extension command contained in it by using just the command's name:

```
!foo [args]
```

You can explicitly load an extension DLL with either of the following commands:

```
!mydll.foo
!load mydll.dll
```

There can be as many as 32 extension DLLs loaded, including the default extension DLL, `kdext???`, where `???` is `x86`, `mip`, or `alp`, depending upon the type of target machine.

When you run `!foo`, WinDbg looks for `foo` in the current extension DLL, which is the last one loaded or the one that you specify with

```
!default mydll
```

If it does not find `foo` in the current extension DLL, WinDbg searches through the loaded extension DLLs, in the order that they were loaded, and runs the first instance of `foo` that it finds.

You can unload the current extension DLL with the following command:

```
!unload
```

The DDK provides source code for a few simple debugger extension samples in the `src\krnl_dbg\kdexts` directory of the DDK tree. Use the files `wdbgexts.h` and `dbgexts.c` (below) as a starting point for writing your own extension DLLs. There are certain requirements for your DLLs:

1. Your extension names must be lowercase: `foo` is good; `Foo` isn't.
2. You must export a function named `WinDbgExtensionDllInit` (refer to the source for `dbgexts.c`, below). When WinDbg loads an extension DLL, it first calls `WinDbgExtensionDllInit` and passes it the following arguments:

Pointer to an extension API structure	<p>This structure contains the callbacks to functions that you can use to do standard operations. For example, to print a string you can use <code>dprintf("string")</code>. Include the header file <code>wdbgexts.h</code> in your extension source and use the <code>#define</code> function definitions from that file. The file <code>wdbgexts.h</code> also defines the parameters to the functions.</p> <p>Save the pointer in a global variable named <code>ExtensionApis</code> (refer to <code>dbgexts.c</code>, below).</p>
Major version	This indicates whether the target system is running a checked build of Windows NT (0x0c) or a free build (0x0f).
Minor version	This is the Windows NT build number of the target system (for example, 801).

3. You must export a function called `ExtensionApiVersion` (refer to the source for `dbgexts.c`, below). WinDbg calls this function and expects back a pointer to `API_VERSION`. The version number of your extension DLL must match the version number of WinDbg. If the numbers don't match, WinDbg will not load the extension DLL.



4. You can optionally export a function called `CheckVersion` (refer to the source for `dbgexts.c`, below). WinDbg calls this function every time you use your DLL. Use it to verify that the version of your DLL matches the target system. To disable version checking, issue the `!noverison` command at the WinDbg prompt.
5. Use the `DECLARE_API` macro defined in `wdbgexts.h` to declare your command functions. The basic format is:

```
DECLARE_API (foo)
{
    code for foo
}
```

The file `wdbgexts.h` defines the following APIs that you can call from your DLL:

API	Description
<u><a href="#">dprintf</a></u>	Prints a formatted string to WinDbg's Command window.
<u><a href="#">GetExpression</a></u>	Returns the value of an expression.
<u><a href="#">GetSymbol</a></u>	Locates the nearest symbol.
<u><a href="#">Disasm</a></u>	Disassembles an instruction.
<u><a href="#">CheckControlC</a></u>	Checks to see if the user pressed CTRL+C.
<u><a href="#">ReadMemory</a></u>	Reads memory from the process being debugged.
<u><a href="#">WriteMemory</a></u>	Writes memory to the process being debugged.
<u><a href="#">GetContext</a></u>	Returns the context of the process being debugged.
<u><a href="#">SetContext</a></u>	Sets the context of the process being debugged.
<u><a href="#">StackTrace</a></u>	Retrieves a stacktrace for the process being debugged.
<u><a href="#">GetKDCContext</a></u>	Gets the current processor number and the total number of processors.
<u><a href="#">ReadControlSpace</a></u>	Reads implementation-specific system data.
<u><a href="#">ReadIoSpace</a></u>	Reads the regular I/O space.
<u><a href="#">WriteIoSpace</a></u>	Writes the regular I/O space.
<u><a href="#">ReadIoSpaceEx</a></u>	Reads the I/O space beyond what <code>ReadIoSpace</code> can read.
<u><a href="#">WriteIoSpaceEx</a></u>	Writes the I/O space beyond what <code>WriteIoSpace</code> can write.
<u><a href="#">ReadPhysical</a></u>	Reads physical memory.
<u><a href="#">WritePhysical</a></u>	Writes physical memory.

WinDbg does a try/except around a call to an extension DLL. Even though they won't crash WinDbg, bugs in your code can still cause it not to work properly, and you will have to quit and restart WinDbg.

#### Source for `dbgexts.c`:

```
/*++
```

```
Copyright (c) 1993 Microsoft Corporation
```

```
Module Name:
```

```
    dbgexts.c
```

```
Abstract:
```

This file contains the minimum required apis for a debugger extension dll.

Environment:

User Mode

--\*/

#include <imagehlp.h>

//

// globals

//

API\_VERSION                      ApiVersion = { 3, 5, API\_VERSION\_NUMBER, 0 };

WINDBG\_EXTENSION\_APIS      ExtensionApis;

USHORT                      SavedMajorVersion;

USHORT                      SavedMinorVersion;

VOID

WinDbgExtensionDllInit(

    PWINDBG\_EXTENSION\_APIS lpExtensionApis,

    USHORT MajorVersion,

    USHORT MinorVersion

)

{

    ExtensionApis = \*lpExtensionApis;

    SavedMajorVersion = MajorVersion;

    SavedMinorVersion = MinorVersion;

    return;

}

VOID

CheckVersion(

    VOID

)

{

    //

    // your check version code goes here

    //

}

LPAPI\_VERSION

ExtensionApiVersion(

    VOID

)

{

    return &ApiVersion;

}



## Building Your Driver

The DDK's build utility uses the settings created by the *setenv.bat* batch file to provide a simple mechanism that can be customized for setting up the build environment. This batch file sets various environment variables that control the compiler and linker options and definitions used when the build utility uses the *makefile.def* file. These include the command-line switches that determine the level of debugging symbols stored in the executable files, and the compiler-specific definitions that cause optional sections of code to be included or excluded.

These files determine the way your driver is built. You can use the default versions of these files, or modify them according to your preferences.

Specifying "checked" on the *setenv* command line sets up the build environment to produce a debugging version of the driver that corresponds to the checked build of Windows NT. Similarly, specifying "free" results in a streamlined version of the driver that corresponds to the free build of Windows NT. When you are building a driver for debugging, build the version (checked or free) that corresponds to the version of the Windows NT system in which it will be tested.

The specific compiler and linker options that are enabled for checked and free builds vary depending on the platform (Intel, MIPS, or Alpha) for which you are building the driver. For example, a checked x86 build using the default versions of *setenv.bat* and *makefile.def* causes the build utility to define the compiler variable `DBG=1`, and to use the following `cl386` compiler switches:

- `/Z7` Produces an object file with line-number and public symbol information.
- `/Oy-` Enables the creation of frame pointers on the call stack.
- `/Odi` Disables optimization.

A free x86 build defines `DBG=0`, omits the `/Z7` compiler switch to reduce the amount of symbolic information in the object files, and uses the `/Oy` switch to enable frame pointer optimization.

To build a free version of your driver with CodeView symbols, which provide for source-stepping, set the environment variables `NTDEBUG` and `NTDEBUGTYPE` as follows:

```
set NTDEBUG=NTSD
set NTDEBUGTYPE=WINDBG
```

To build a free version of your driver with COFF symbols, which do not provide for source-stepping, set the variables to nothing:

```
set NTDEBUG=
set NTDEBUGTYPE=
```

To build a checked version of your driver, set the variables as follows:

```
set NTDEBUG=NTSD
set NTDEBUGTYPE=WINDBG
```

To disable optimization, set the environment variable `MSC_OPTIMIZATION` as follows:

```
set MSC_OPTIMIZATION=/Odi
```



## **dprintf**

### **Syntax**

```
VOID dprintf(char *format [, argument] ...);
```

### **Parameters**

*format*

The format string to print, as in printf.

*argument*

Arguments for the format string, as in printf.

### **Description**

Prints a formatted string to the WinDbg Command window. Works like printf.



## GetExpression

### Syntax

```
DWORD GetExpression(char *expression);
```

### Parameters

*expression*

The expression to evaluate.

### Description

Returns the value of *expression*, using WinDbg's C/C++ expression evaluator. Equivalent to using the ? command in the WinDbg Command window.



## GetSymbol

### Syntax

```
VOID GetSymbol(LPVOID address, PCHAR buffer, LPDWORD lpdwDisplacement);
```

### Parameters

*address*

The address for which you want WinDbg to locate the nearest symbol.

*buffer*

Returns the name of the symbol found.

*lpdwDisplacement*

The displacement from the beginning of the symbol.

### Description

Locates the symbol nearest to *address*.



## Disasm

### Syntax

```
DWORD Disasm(LPDWORD lpOffset, LPSTR lpBuffer, BOOL  
fShowEffectiveAddress);
```

### Parameters

*lpOffset*

Pointer to the instruction to be disassembled.

*lpBuffer*

Returns the disassembled instruction.

*fShowEffectiveAddress*

Controls whether or not to print the effective address, if appropriate.

### Description

Disassembles the instruction pointed to by *lpOffset* and places the printable string into *lpBuffer*.



## **CheckControlC**

### **Syntax**

```
BOOL CheckControlC (VOID) ;
```

### **Parameters**

There are no parameters.

### **Description**

Checks to see if the user pressed CTRL+C. Use CheckControlC in all loops to allow the user to press CTRL+C to end long processes.





## ReadMemory

### Syntax

```
BOOL ReadMemory(DWORD offset, LPVOID lpBuffer, DWORD cb, LPDWORD  
lpcbBytesRead);
```

### Parameters

*offset*

The base address in the process being debugged of the memory to be read.

*lpBuffer*

Pointer to the buffer to receive the memory read.

*cb*

The number of bytes that you want ReadMemory to read.

*lpcbBytesRead*

The actual number of bytes that ReadMemory transferred into the buffer. This parameter is optional; if it is NULL, it is ignored.

### Description

ReadMemory works like the Win32 API ReadProcessMemory. It reads memory from the process being debugged. The entire area to be read must be accessible, or the operation fails. If the function succeeds, the return value is TRUE; otherwise, it is FALSE.



## WriteMemory

### Syntax

```
BOOL WriteMemory(DWORD offset, LPVOID lpBuffer, DWORD cb, LPDWORD  
lpcbBytesWritten);
```

### Parameters

*offset*

The base address in the process being debugged of the memory to be written.

*lpbuffer*

Pointer to the buffer that contains the data to be written.

*cb*

The number of bytes that you want WriteMemory to write.

*lpcbBytesWritten*

The number of bytes that WriteMemory transferred from the buffer. This parameter is optional; if it is NULL, it is ignored.

### Description

WriteMemory works like the Win32 API WriteProcessMemory. It writes memory to the process being debugged. The entire area to be written must be accessible, or the operation fails. If the function succeeds, the return value is TRUE; otherwise, it is FALSE.



## GetContext

### Syntax

```
BOOL GetContext(LPCONTEXT lpContext, DWORD cbSizeOfContext);
```

### Parameters

*lpContext*

Points to the address of a context structure that receives the appropriate context of the process being debugged. The context structure is highly machine-specific. Two versions of the context structure currently exist, one version for x86 processors and another for MIPS processors.

*cbSizeOfContext*

The size of the context structure that you want to read.

### Description

GetContext is similar to the Win32 API GetThreadContext. It returns the context of the process being debugged. If the function succeeds, the return value is TRUE; otherwise, it is FALSE.



## SetContext

### Syntax

```
BOOL SetContext(LPCONTEXT lpContext, DWORD cbSizeOfContext);
```

### Parameters

*lpContext*

Points to the address of a context structure that contains the context to be set for the process being debugged. The context structure is highly machine-specific. Two versions of the context structure currently exist, one version for x86 processors and another for MIPS processors.

*cbSizeOfContext*

The size of the context structure that you want to set.

### Description

SetContext is similar to the Win32 API SetThreadContext. It sets the context of the process being debugged. If the function succeeds, the return value is TRUE; otherwise, it is FALSE.



## StackTrace

### Syntax

```
DWORD StackTrace(DWORD FramePointer, DWORD StackPointer, DWORD  
ProgramCounter, PEXTSTACKTRACE StackFrames, DWORD Frames);
```

### Parameters

*FramePointer*

Set to an initial value of zero.

*StackPointer*

Set to an initial value of zero.

*ProgramCounter*

Set to an initial value of zero.

*StackFrames*

Pointer to a buffer that is large enough to hold the number of stack frames specified by *Frames*.

*Frames*

The number of frames that you want to read into the buffer.

### Description

Retrieves a stacktrace for the process being debugged. Returns the number of frames read into the buffer pointed to by *StackFrames*.



## GetKDContext

### Syntax

```
GetKDContext(DWORD ppi);
```

### Parameters

*ppi*

*ppi* is the address of the following structure:

```
typedef struct _tagPROCESSORINFO {  
    USHORT   Processor;           // current processor  
    USHORT   NumberProcessors;    // total number of processors  
} PROCESSORINFO, *PPROCESSORINFO;
```

### Description

GetKDContext returns the total number of processors and the number of the current processor in the structure pointed to by *ppi*.



## ReadControlSpace

### Syntax

```
ReadControlSpace(ULONG processor, ULONG address, BYTE buf, ULONG size);
```

### Parameters

*processor*

The number of the processor whose control space you want to read.

*address*

The address of the control space.

*buf*

The address of an array of bytes to hold the control space data.

*size*

The number of bytes in the array pointed to by *buf*.

### Description

Reads the processor-specific control space into the array pointed to by *buf*.



## ReadIoSpace

### Syntax

```
ReadIoSpace(ULONG address, ULONG data, ULONG size);
```

### Parameters

*address*

The I/O address to read from.

*data*

The address of a variable to hold the data read (must be at least *size* bytes long).

*size*

The address of a variable that contains the number of bytes to read (1, 2, or 4 only). After the data is read, *size* will contain the number of bytes actually read.

### Description

Reads the system I/O locations.





## WriteIoSpace

### Syntax

```
WriteIoSpace(ULONG address, ULONG data, ULONG size);
```

### Parameters

*address*

The I/O address to write to.

*data*

The address of a variable that holds the data to write (must be at least *size* bytes long).

*size*

The address of a variable that contains the number of bytes to write (1, 2, or 4 only). After the data is written, *size* will contain the number of bytes actually written.

### Description

Writes the system I/O locations.



## ReadIoSpaceEx

### Syntax

```
ReadIoSpaceEx(ULONG address, ULONG data, ULONG size, ULONG interfacetype,  
              ULONG busnumber, ULONG addressspace);
```

### Parameters

*address*

The I/O address to read from.

*data*

The address of a variable to hold the data read (must be at least *size* bytes long).

*size*

The address of a variable that contains the number of bytes to read (1, 2, or 4 only). After the data is read, *size* will contain the number of bytes actually read.

*interfacetype*

The type of interface on which the extended I/O space exists. Values include Isa, Eisa, MicroChannel, etc. (see ntddk.h).

*busnumber*

The number of the bus on which the extended I/O space exists. This is typically 0, unless there is more than one bus of a given type.

*addressspace*

This is typically 1.

### Description

ReadIoSpaceEx is an extended version of ReadIoSpace, and is supported on Alpha processors only. It reads not only the system I/O locations, but also I/O locations on a bus. ReadIoSpace works like ReadIoSpaceEx, except that it defaults *interfacetype* to Isa, *busnumber* to 0, and *addressspace* to 1.



## WriteIoSpaceEx

### Syntax

```
WriteIoSpaceEx(ULONG address, ULONG data, ULONG size, ULONG interfacetype,  
               ULONG busnumber, ULONG addressspace);
```

### Parameters

*address*

The I/O address to write to.

*data*

The address of a variable that holds the data to write (must be at least *size* bytes long).

*size*

The address of a variable that contains the number of bytes to write (1, 2, or 4 only). After the data is written, *size* will contain the number of bytes actually written.

*interfacetype*

The type of interface on which the extended I/O space exists. Values include Isa, Eisa, MicroChannel, etc. (see ntddk.h).

*busnumber*

The number of the bus on which the extended I/O space exists. This is typically 0, unless there is more than one bus of a given type.

*addressspace*

This is typically 1.

### Description

WriteIoSpaceEx is an extended version of WriteIoSpace, and is supported on Alpha processors only. It writes not only the system I/O locations, but also I/O locations on a bus. WriteIoSpace works like WriteIoSpaceEx, except that it defaults *interfacetype* to Isa, *busnumber* to 0, and *addressspace* to 1.



## ReadPhysical

### Syntax

```
ReadPhysical(PHYSICAL_ADDRESS address, ULONG buf, BYTE size, ULONG sizer);
```

### Parameters

*address*

The physical address to read.

*buf*

The address of an array of bytes to hold the data read.

*size*

The number of bytes to read.

*sizer*

The address of a variable to receive the number of bytes actually read.

### Description

Reads physical memory.



## WritePhysical

### Syntax

```
WritePhysical(PHYSICAL_ADDRESS address, ULONG buf, BYTE size, ULONG sizew);
```

### Parameters

*address*

The physical address to write.

*buf*

The address of an array of bytes to hold the data written.

*size*

The number of bytes to write.

*sizew*

The address of a variable to receive the number of bytes actually written.

### Description

Writes physical memory.

