SoftICE™ Command Reference

Windows NT™
Windows® 98
Windows® 95
Windows® 3.1
DOS

Compuware® NuMega™

July 1998

Part number 0000-55-2750

This Software License Agreement is not applicable if You have a valid Compuware License Agreement and have licensed this Software under a Compuware Product Schedule.

Software License Agreement

Please Read This License Carefully

You are purchasing a license to use Compuware Corporation Software. The Software is the property of Compuware Corporation and/or its licensors, is protected by intellectual property laws, and is provided to You only on the license terms set forth below. This Agreement does not transfer title to the intellectual property contained in the Software. Compuware reserves all rights not expressly granted to you. Opening the package and/or using the Software indicates your acceptance of these terms. If you do not agree to all of the terms and conditions, or if after using the Software you are dissatisfied, return the Software, manuals and any copies within thirty (30) days of purchase to the party from whom you received it for a refund, subject in certain cases to a restocking fee.

Title and Proprietary Rights: You acknowledge and agree that the Software is proprietary to Compuware and/or its licensors, and is protected under the laws of the United States and other countries. You further acknowledge and agree that all rights, title and interest in and to the Software, including intellectual property rights, are and shall remain with Compuware and/or its licensors. Unauthorized reproduction or distribution is subject to civil and criminal penalties.

Use Of The Software: Compuware Corporation ("Compuware") grants a single individual (“You”) the limited right to use the Compuware software product(s) and user manuals included in the package with this license ("Software"), subject to the terms and conditions of this Agreement. You agree that the Software will be used solely for your internal purposes, and that at any one time, the Software will be installed on a single computer only. If the Software is installed on a network system or on a computer connected to a file server or other system that physically allows shared access to the Software, You agree to provide technical or procedural methods to prevent use of the Software by more than one individual. Individuals other than You may not have access to the Software even at different times.

One machine-readable copy of the Software may be made for BACK UP PURPOSES ONLY, and the copy shall display all proprietary notices, and be labeled externally to show that the back-up copy is the property of Compuware, and that its use is subject to this License. Documentation may not be copied in whole or part.

You may not use, transfer, assign, export or in any way permit the Software to be used outside of the country of purchase, unless authorized in writing by Compuware.

Except as expressly provided in this License, You may not modify, reverse engineer, decompile, disassemble, distribute, sub-license, sell, rent, lease, give or in any way transfer, by any means or in any medium, including telecommunications, the Software. You will use your best efforts and take all reasonable steps to protect the Software from unauthorized use, copying or dissemination, and will maintain all proprietary notices intact.

Government Users: With respect to any acquisition of the Software by or for any unit or agency of the United States Government, the Software shall be classified as "commercial computer software", as that term is defined in the applicable provisions of the Federal Acquisition Regulation (the "FAR") and supplements thereto, including the Department of Defense (DoD) FAR Supplement (the "DFARS"). If the Software is supplied for use by DoD, the Software is delivered subject to the terms of this Agreement and either (i) in accordance with DFARS 227.7202-1(a) and 227.7202-3(a), or (ii) with restricted rights in accordance with DFARS 252.227-7013(c)(1)(ii) (OCT 1988), as applicable. If the Software is supplied for use by a Federal agency other than DoD, the Software is restricted computer software delivered subject to the terms of this Agreement and (i) FAR 12.212(a); (ii) FAR 52.227-19; or (iii) FAR 52.227-14(ALT III), as applicable. Licensor: Compuware Corporation, 31440 Northwestern Highway, Farmington Hills, Michigan 48334.

Limited Warranty and Remedy: Compuware warrants the Software media to be free of defects in workmanship for a period of ninety (90) days from purchase. During this period, Compuware will replace at no cost any such media returned to Compuware, postage prepaid. This service is Compuware's sole liability under this warranty. COMPUWARE DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN THAT EVENT, ANY IMPLIED WARRANTIES ARE LIMITED IN DURATION TO THIRTY (30) DAYS FROM THE DELIVERY OF THE SOFTWARE. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

Infringement of Intellectual Property Rights: In the event of an intellectual property right claim, Compuware agrees to indemnify and hold You harmless provided You give Compuware prompt written notice of such claim, permit Compuware to defend or settle the claim and provide all reasonable assistance to Compuware in defending or settling the claim. In the defense or settlement of such claim, Compuware may obtain for You the right to continue using the Software or replace or modify the Software so that it avoids such claim, or if such remedies are not reasonably available, accept the return of the infringing Software and provide You with a pro-rata refund of the license fees paid for such Software based on a three (3) year use period.

Limitation of Liability: YOU ASSUME THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE. IN NO EVENT WILL COMPUWARE BE LIABLE TO YOU OR TO ANY THIRD PARTY FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING BUT NOT LIMITED TO, LOSS OF USE, DATA, REVENUES OR PROFITS, ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT OR THE USE, OPERATION OR PERFORMANCE OF THE SOFTWARE, WHETHER SUCH LIABILITY ARISES FROM ANY CLAIM BASED UPON CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, AND WHETHER OR NOT COMPUWARE OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT SHALL COMPUWARE BE LIABLE TO YOU FOR AMOUNTS IN EXCESS OF PURCHASE PRICE PAID FOR THE SOFTWARE.

Terms and Termination

This License Agreement shall be effective upon your acceptance of this Agreement and shall continue until terminated by mutual consent, or by election of either You or Compuware in case of the other’s unremediated material breach. In case of any termination of the Agreement, you will immediately return to Compuware the Software that You have obtained under this Agreement and will certify in writing that all copies of the Software have been returned or erased from the memory of your computer or made non-readable.

General: This License is the complete and exclusive statement of the parties' agreement. Should any provision of this License be held to be invalid by any court of competent jurisdiction, that provision will be enforced to the maximum extent permissible and the remainder of the License shall nonetheless remain in full force and effect. This Agreement shall be governed by the laws of the State of Michigan and the United States of America.

Contents

. 2

? 3

A 4

ACTION 6

ADDR 7

ADDR 10

ALTKEY 12

ALTSCR 13

ANSWER 14

BC 16

BD 17

BE 18

BH 19

BL 21

BMSG 22

BPE 24

BPINT 25

BPINT 27

BPIO 29

BPM 32

BPR 36

BPRW 39

BPT 41

BPX 42

BSTAT 45

C 47

CLASS 48

CLS 51

CODE 52

COLOR 53

CPU 55

CR 58

CSIP 59

D 61

DATA 63

DEVICE 64

DEX 66

DIAL 67

DRIVER 69

E 71

EC 73

EXIT 74

EXP 75

F 78

FAULTS 79

FIBER 80

FILE 81

FKEY 82

FOBJ 84

FLASH 86

FORMAT 87

G 88

GDT 89

GENINT 91

H 92

HBOOT 93

HEAP 94

HEAP32 97

HEAP32 100

HERE 105

HWND 106

HWND 109

I 113

I1HERE 114

I3HERE 115

IDT 116

IRP 118

LDT 121

LHEAP 123

LINES 125

LOCALS 126

M 127

MACRO 128

MAP32 132

MAPV86 135

MOD 137
MOD 139
NTCALL 142
O 144
OBJDIR 145
OBJTAB 147
P 149
PAGE 150
PAUSE 155
PCI 156
PEEK 157
PHYS 158
POKE 159
Print Screen Key 160
PRN 161
PROC 162
QUERY 168
R 173
RS 175
S 176
SERIAL 178
SET 181
SHOW 183
SRC 184
SS 185
STACK 186
SYM 189
SYMLOC 191
T 193
TABLE 194
TABS 196
TASK 197
THREAD 199
THREAD 201
TRACE 204
TSS 205
TYPES 207
U 208
VCALL 210
VER 212
VM 213
VXD 216
VXD 218
WATCH 220
WC 222
WD 223
WF 224
WHAT 226
WL 227
WMSG 228
WR 229
WW 230
X 231
XFRAME 232
XG 234
XP 235
XRSET 236
XT 237
ZAP 238

You will find it a very good practice always
to verify your references, sir!

◊ **Dr. Routh**

SoftICE Commands

The *SoftICE Command Reference* is for use with the following operating systems:

- Windows 3.1
- Windows 95
- Windows 98
- Windows NT

The commands are listed in alphabetical order and contain the following information:

Note: Some commands list support for Windows 98. These commands do not consistently work the same way they do on Windows 95 or Windows NT. Do not assume similarity.

Command name — **OBJDIR** Windows 98, *Windows NT* *System Information*

Operating systems

Displays objects in a Windows NT Object Manager's object directory.

Syntax and parameters — **Syntax** **OBJDIR [object-directory-name]**

Command use — **Use** Use the OBJDIR command to display named objects within the Object Manager's object directory. Using OBJDIR with no parameters displays the named objects within the root object directory.

Sample output — **Output** The OBJDIR command displays the following information:

Object	Address of the object body
ObjHdr	Address of the object header
Name	Name of the object
Type	Windows NT-defined data type of the object

Example(s) — **Example** Abbreviated sample output of the OBJDIR command listing objects in the Device object directory follows:

OBJDIR device
Directory of \Device at FD8E7F30

Object	ObjHdr	Name	Type
FD8CC750	FD8CC728	Beep	Device
FD89A030	FD89A008	Nwlnklpx	Device
FD889150	FD889128	Netbios	Device

Lists related commands — **See Also** OBJTAB

These sections also include any operating system specific information.

Type of SoftICE command:

- Breakpoints and Watches
- Customization
- Display/Change Memory
- Flow Control
- I/O Port
- Manipulating Breakpoints
- Miscellaneous
- Mode Control
- Symbol/Source
- System Information
- Window Control

.

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Locate the current instruction in the Code window.

Syntax

```
.
```

Use

When the Code window is visible, the `.` (Dot) command makes the instruction at the current CS:EIP visible and highlights it.

For Windows 95 and Windows NT

The command switches contexts back to the original context that SoftICE popped up in.

?

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Evaluate an expression.

Syntax

For Windows 3.1

? [*command* | *expression*]

For Windows 95 and Windows NT

? *expression*

Use

For Windows 3.1

Under Windows 3.1, the parameter you supply to the ? command determines whether help is displayed or an expression is evaluated. If you specify a command, ? displays detailed information about the command, including the command syntax and an example. If you specify an expression, the expression is evaluated, and the result is displayed in hexadecimal, decimal, signed decimal (only if < 0), and ASCII.

For Windows 95 and Windows NT

Under Windows 95 and Windows NT, the ? command only evaluates expressions. (Refer to *H* on page 92 for information about getting help under Windows 95 and Windows NT.)

To evaluate an expression enter the ? command followed by the expression you want to evaluate. SoftICE displays the result in hexadecimal, decimal, signed decimal (only if < 0), and ASCII.

Example

The following command displays the hexadecimal, decimal, and ASCII representations of the value of the expression 10*4+3.

```
:? 10*4+3
00000043 0000000067  "C"
```

See Also

H

A

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Assemble code.

Syntax

A [***address***]

Use

Use the SoftICE assembler to assemble instructions directly into memory. The assembler supports the standard Intel 80x86 instruction set.

If you do not specify the address, assembly occurs at the last address where instructions were assembled. If you have not entered the A command before and did not specify the address, the current CS:EIP address is used.

The A command enters the SoftICE interactive assembler. An address displays as a prompt for each assembly line. After you type an assembly language instruction and press Enter, the instructions assemble into memory at the specified address. Type instructions in the standard Intel format. To exit assembler mode, press Enter at an address prompt.

If the address range in which you are assembling instructions is visible in the Code window, the instructions change interactively as you assemble.

The SoftICE assembler supports the following instruction sets:

- For Windows 3.1: 386, Floating Point
- For Windows 95 and Windows NT: 386, 486, Pentium, Pentium Pro, all corresponding numeric coprocessor instruction sets, and MMX instruction sets

SoftICE also supports the following special syntax:

- Enter USE16 or USE32 on a separate line to assemble subsequent instructions as 16-bit or 32-bit, respectively. If you do not specify USE16 or USE32, the default is the same as the mode of the current CS register.
- Mnemonic followed by a list of bytes and/or quoted strings separated by spaces or commas.
- RETF mnemonic represents a far return.
- Use WORD PTR, BYTE PTR, DWORD PTR, and FWORD PTR to determine data size, if there is no register argument.

  Example: `MOV BYTE PTR ES:[1234.],1`
- Use FAR and NEAR to explicitly assemble far and near jumps and calls. If you do not specify either, the default is NEAR.
- Place operands referring to memory locations in square brackets.

  Example: `MOV AX,[1234]`

For Windows NT

Any changes you make to 32-bit code are “sticky.” This means they remain in place even if you load or reload the module you change. To remove the changes, do one of the following: restart Windows NT, flush the memory image from the cache, or modify the module.

Example

When you use the following command:

```
A CS:1234
```

the assembler prompts you for assembly instructions. Enter all instructions and press Enter at the address prompt. The assembler assembles the instructions beginning at offset 1234h within the current code segment.

ACTION

Windows 3.1 *Mode Control*

Set action after breakpoint is reached.

Syntax

ACTION [nmi|int1|int3|here|*interrupt-number*|*debugger-name*]

interrupt-number	Valid interrupt number between 0 and 5Fh.
debugger-name	Module name of the Windows application debugger to gain control of on a SoftICE breakpoint.

Use

The ACTION command determines where to give control when breakpoint conditions are met. In most cases, you can use ACTION to pass control to an application debugger you are using in conjunction with SoftICE. Use the HERE parameter to return to SoftICE when break conditions have been met. Use the NMI, INT1, and INT3 parameters as alternatives for activating DOS debuggers when break conditions are met. Use debugger-name to activate Windows debuggers. To find the module name of the debugger, use the MOD command.

If you specify debugger-name, an INT 0 triggers the Windows debugger. SoftICE ignores breakpoints that the Windows debugger causes if the debugger accesses memory covered by a memory location or range breakpoint. When SoftICE passes control to the Windows debugger with an INT 0, the Windows debugger responds as if a divide overflow occurred and displays a message. Ignore this message because the INT 0 was not caused by an actual divide overflow.

Note: The ACTION command is obsolete under Windows 95 and Windows NT.

Example

When using SoftICE with the following products, use the corresponding command:

Product	SoftICE Command
CodeView for DOS	**ACTION nmi** *Note:* SoftICE generates a non-maskable interrupt when break conditions are met. This gives control to CodeView for DOS.
CodeView for Windows	**ACTION cvw**
Borland's Turbo Debugger for Windows	**ACTION tdw**
Multiscope's Debugger for Windows	**ACTION rtd**

See Also

Refer to setting breakpoints in *Using SoftICE*.

ADDR

Windows 95, Windows 98 *System Information*

Display or switch to address context.

Syntax

ADDR [***context-handle*** | ***process-name***]

context-handle	Address context handle.
process-name	Name of a process.

Use

To be able to view the private address space for an application process, set the current address context within SoftICE to that of the application by providing an address context-handle or the process-name as the first parameter to the ADDR command. To view information on all currently active contexts, use ADDR with no parameters. The first address context listed is the current address context.

To use ADDR with Windows NT, refer to ADDR on page 10.

For each address context, SoftICE prints the following information:

- address context handle
- address of the private page table entry array (PGTPTR) of the context
- number of entries that are valid in the PGTPTR array
- starting and ending linear addresses represented by the context
- address of the mutex object used to control access to the context's page tables
- name of the process that owns the context.

When you use the ADDR command with an address context parameter, SoftICE switches address contexts the same way as Windows does.

When switching address contexts, Windows 95 copies all entries in the new context's PGTPTR array to the page directory (pointed at by the CR3 register). A context switch affects the addressing of the lower 2GB of memory from linear address 0 to 7FFFFFFFh. Each entry in a PGTPTR array is a page directory entry which points at a page table that represents 4MB of memory. There can be a maximum of 512 entries in the PGTPTR array to represent the full 2GB. If there are less than 512 entries in the array, the rest of the entries in the page directory are set to invalid values.

When running more than one instance of an application, the same owner name appears in the address context list more than once. If you specify an owner name as a parameter, SoftICE always selects the first address context with a matching name in the list. To switch to the address context of a second or third instance of an application, provide an address context-handle to the ADDR command.

Note: If SoftICE pops up when the System VM (VM 1) is not the current VM, it is possible for context owner information to be paged out and unavailable. In these cases no owner information displays.

Output

For each context or process, the following information displays.

Handle	Address of the context control block. This is the handle that is passed in VxD calls that require a context handle.
Pgtptr	Address of an array of page table addresses. Each entry in the array represents a page table pointer. When address contexts switch, the appropriate location in the page directory receives a copy of this array.
Tables	Number of entries in the PGTPTR array. Not all entries contain valid page directory entries. This is only the number of entries reserved.
MinAddr	Minimum linear address of the address context.
MaxAddr	Maximum address of the address context.
Mutex	Mutex handle used when VMM manipulates the page tables for the context.
Owner	Name of the first process that uses this address context.

Example

The following command displays all currently active address contexts. The context on the top line of the display is the context that SoftICE popped up in. To switch back to this at any time, use the . (DOT) command. When displaying information on all contexts, one line is highlighted, indicating the current context within SoftICE. When displaying data or disassembling code, the highlighted context is the one you see.

.:ADDR

The current context is highlighted.

Handle	PGTPTR	Tables	Min Addr	Max Addr	Mutex	Owner
C1068D00	C106CD0C	0200	00400000	7FFFF000	C0FEC770	WINWORD
C104E214	C1068068	0200	00400000	7FFFF000	C1063DBC	Rundll32
C105AC9C	C0FE5330	0002	00400000	7FFFF000	C0FE5900	QUICKRES
C1055EF8	C105CE8C	0200	00400000	7FFFF000	C105C5EC	Ibserver
C1056D10	**C10571D4**	**0200**	**00400000**	**7FFFF000**	**C1056D44**	**Mprexe**

Handle	PGTPTR	Tables	Min Addr	Max Addr	Mutex	Owner
C10D900C	C10D9024	0002	00400000	7FFFF000	C10D9050	
C10493E8	C10555FC	0004	00400000	7FFFF000	C0FE6460	KERNEL32
C1055808	C105650C	0200	00400000	7FFFF000	C105583C	MSGSRV32
C10593CC	C1059B78	0200	00400000	7FFFF000	C105908C	Explorer
C106AE70	C106DD10	0200	00400000	7FFFF000	C10586F0	Exchng32
C106ABC4	C106ED04	0200	00400000	7FFFF000	C106CA4C	Mapisp32

See Also

For Windows NT, refer to *ADDR* on page 10.

ADDR

Windows NT *System Information*

Display or switch to an address context.

Syntax

ADDR [*process-name* | *process-id* | KPEB]

KPEB	Kernel Process Environment Block.

Use

Use the ADDR command to both display and change address contexts within SoftICE so that process-specific data and code can be viewed. Using ADDR with no parameters displays a list of all address contexts.

If you specify a parameter, SoftICE switches to the address context belonging to the process with that name, identifier, or process control block address.

To use ADDR with Windows 95, refer to ADDR on page 7.

If you switch to an address context that contains an LDT, SoftICE sets up the LDT with the correct base and limit.

All commands that use an LDT only work when the current SoftICE context contains an LDT. LDTs are *never* global under Windows NT.

Under low memory conditions, Windows NT starts swapping data to disk, including inactive processes, parts of the page directory, and page tables. When this occurs, SoftICE may not be able obtain the information necessary to switch to contexts that rely on this information. SoftICE indicates this by displaying the message `swapped` in the CR3 field of the process or displaying an error message if an attempt is made to switch to the context of the process.

When displaying information about all contexts, one line is highlighted, indicating the current context within SoftICE. When displaying data or disassembling code, the highlighted context is the one you see.

An `*` (asterisk) precedes one line of the display, indicating the process that was active when SoftICE popped up. Use the . (DOT) command to switch contexts back to this context at any time.

Output

For each context or process, the following information is shown:

CR3	Physical address of the page directory that is placed into the CR3 register on a process switch to the process.
LDT	If the process has an LDT, this field has the linear base address of the LDT and the limit field for the LDT selector. All Windows NT processes that have an LDT use the same LDT selector. For process switches, Windows NT sets the base and limit fields of this selector.

KPEB	Linear address of the Kernel Process Environment Block for the process.
PID	Process ID. Each process has a unique ID.
NAME	Name of the process.

Example

The following example shows the ADDR command being used without parameters to display all the existing contexts.

:ADDR

CR3	LDT Base:Limit	KPEB	PID	NAME
00030000		FD8EA920	0002	System
011FB000		FD8CD880	0013	smss
017A5000		FD8BFB60	0016	csrss
01B69000		FD8BADE0	001B	winlogon
01CF3000		FD8B6B40	0027	services
01D37000		**FD8B5760**	**0029**	**lsass**
00FFA000		FD8A8AE0	0040	spoolss
009A5000		FD89F7E0	002B	nddeagnt
00AA5000		FD89CB40	004A	progman
006D2000	E115F000:FFEF	FD899DE0	0054	ntvdm
00837000		FD896D80	0059	CLOCK
00C8C000		FD89C020	0046	scm
00387000		FD89E5E0	004E	4NT
*0121C000	E1172000:0187	FD88CCA0	0037	ntvdm
00030000		8013DD50	0000	Idle

See Also

For Windows 95, refer to *ADDR* on page 7.

PROC

ALTKEY

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Set an alternate key sequence to invoke SoftICE.

Syntax

ALTKEY [**Alt** ***letter*** | **Ctrl** ***letter***]

letter Any letter (A through Z).

Use

Use the ALTKEY command to change the key sequence (default key Ctrl-D) for popping up SoftICE. Occasionally another program may conflict with the hot key sequence. You can change the key sequence to either of the following sequences:

`Ctrl + letter`

or

`Alt + letter`

If you do not specify a parameter, the current hot key sequence displays.

To change the hot key sequence every time you run SoftICE, Configure SoftICE in the SoftICE Loader to place the ALTKEY command in the SoftICE initialization string.

Example

To specify that the key sequence Alt-Z pop up the SoftICE screen, use the following command:

`ALTKEY alt z`

ALTSCR

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Display SoftICE on an alternate screen.

Syntax

`ALTSCR [on | off]`

Use

Use the ALTSCR command to redirect the SoftICE output from the default screen to an alternate monochrome monitor.

ALTSCR requires the system to have two monitors attached. The alternate monitor should be a monochrome monitor in a character mode (the default mode).

The default setting is ALTSCR mode OFF.

Hint: To change the SoftICE display screen every time you run SoftICE, place the ALTSCR command in the Initialization string within your SoftICE configuration settings. Refer to Chapter 8, “Customizing SoftICE” in the *Using SoftICE* guide.

In the SoftICE program group, use Video Setup to select the monochrome monitor. SoftICE automatically starts out in monochrome mode making the ALTSCR command unnecessary. Also use this setting if you are experiencing video problems even when ALTSCR ON is in the initialization string.

For Windows 95

You can also start WINICE with the /M parameter to bypass the initial VGA programming and force SoftICE to the alternate monochrome screen. This is useful if your video board experiences conflicts with the initial programming.

Example

To redirect screen output to the alternate monitor, use the following command:

```
ALTSCR on
```

ANSWER

Windows 95, Windows 98, Windows NT *Customization*

Auto-answer and redirect console to modem.

Syntax

ANSWER [**on** [***com-port***] [***baud-rate***] [***i=init***] | **off**]

com-port	If no com-port is specified it uses COM1.
baud-rate	Baud-rate to use for modem communications. The default is 38400. The rates include 1200, 2400, 4800, 9600, 19200, 23040, 28800, 38400, 57000, 115000.
i=init	Optional modem initialization string.

Use

The ANSWER command allows SoftICE to answer an incoming call and redirect all output to a connecting PC running the SERIAL.EXE program in dial mode. After the command is executed, SoftICE listens for incoming calls on the specified com-port while the machine continues normal operation. Incoming calls are generated by the SERIAL.EXE program on a remote machine.

You can place a default ANSWER initialization string in the SoftICE configuration settings. Refer to Chapter 8, "Customizing SoftICE" in the *Using SoftICE* guide.

When SoftICE detects a call being made after the ANSWER command has been entered, it pops up and indicates that it is making a connection with a remote machine, then pops down. The local machine appears to be hung while a remote connection is in effect.

The ANSWER command can be cancelled at any time with ANSWER OFF. This stops SoftICE from listening for incoming calls.

Example

The following is an example of the ANSWER command. SoftICE first initializes the modem on com-port 2 with the string "atx0," and then returns control to the command prompt. From that point on it answers calls made on the modem and attempts to connect at a baud rate of 38400bps.

```
ANSWER on 2 38400 i=atx0
```

The following is an example of a default ANSWER initialization string statement in your SoftICE configuration settings. With this statement in place, SoftICE always initializes the modem specified in ANSWER commands with “atx0,” unless the ANSWER command explicitly specifies an initialization string.

```
ANSWER=atx0
```

See Also

SERIAL

BC

Windows 3.1, Windows 95, Windows 98, Windows NT *Manipulating Breakpoints*

Clear one or more breakpoints.

Syntax

BC *list* | *

list	Series of breakpoint indexes separated by commas or spaces.
*	Clears all breakpoints.

Example

To clear all breakpoints, use the command:

```
BC *
```

To clear breakpoints 1 and 5, use the command:

```
BC 1 5
```

If you use the BL command (list breakpoints), the breakpoint list will be empty until you define more breakpoints.

BD

Windows 3.1, Windows 95, Windows 98, Windows NT *Manipulating Breakpoints*

Disable one or more breakpoints.

Syntax

BD *list* | *

list	Series of breakpoint indexes separated by commas or spaces.
*	Disables all breakpoints.

Use

Use the BD command to temporarily deactivate breakpoints. Reactivate the breakpoints with the BE command (enable breakpoints).

To tell which of the breakpoints are disabled, list the breakpoints with the BL command. A breakpoint that is disabled has an * (asterisk) after the breakpoint index.

Example

To disable breakpoints 1 and 3, use the command:

```
BD 1 3
```

BE

Windows 3.1, Windows 95, Windows 98, Windows NT *Manipulating Breakpoints*

Enable one or more breakpoints.

Syntax

BE *list* | *

list	Series of breakpoint indexes separated by commas or spaces.
*	Enables all breakpoints.

Use

Use the BE command to reactivate breakpoints that you deactivated with the BD command (disable breakpoints).

Note: You automatically enable a breakpoint when you first define it or edit it.

Example

To enable breakpoint 3, use the command:

```
BE 3
```

BH

Windows 3.1, Windows 95, Windows 98, Windows NT *Manipulating Breakpoints*

List and/or select previously set breakpoints from the breakpoint history.

Syntax

BH

Use

Use the BH command to recall breakpoints that you set in both the current and previous SoftICE sessions. All saved breakpoints display in the Command window and can be selected using the following keys:

UpArrow	Positions the cursor one line up. If the cursor is on the top line of the Command window, the list scrolls.
DownArrow	Positions the cursor one line down. If the cursor is on the bottom line of the Command window, the list scrolls.
Insert	Selects the breakpoint at the current cursor line, or deselects it if already selected.
Enter	Sets all selected breakpoints.
Esc	Exits breakpoint history without setting any breakpoints.

SoftICE saves the last 32 breakpoints.

For Windows 3.1 and Windows 95

Each time Windows exits normally, these breakpoints are written to the WINICE.BRK file in the same directory as WINICE.EXE. Every time SoftICE is loaded, it reads the breakpoint history from the WINICE.BRK file.

For Windows 95

IF you choose to configure Windows 95 to load SoftICE before WIN.COM by appending \siw95\winice.exe to the end of your AUTOEXEC.BAT, Windows 95 does not return control to SoftICE when it shuts down unless you set the BootGUI option in MSDOS.SYS to BootGUI=0. If this option is set to BootGUI=1, SoftICE does not save the break-point history file. Refer to Chapter 2, "Installing SoftICE," in the *Using SoftICE* manual for more information about configuring when SoftICE loads.

For Windows NT

Breakpoints are written to the WINICE.BRK file in the *SYSTEMROOT*\SYSTEM32 \DRIVERS directory.

Example

To select any of the last 32 breakpoints from current and previous SoftICE sessions, use the command:

BH

BL

Windows 3.1, Windows 95, Windows 98, Windows NT *Manipulating Breakpoints*

List all breakpoints.

Syntax

BL

Use

The BL command displays all breakpoints that are currently set. For each breakpoint, BL lists the breakpoint index, breakpoint type, breakpoint state, and any conditionals or breakpoint actions.

The state of a breakpoint is either enabled or disabled. If you disable the breakpoint, an * (asterisk) appears after its breakpoint index. If SoftICE is activated due to a breakpoint, that breakpoint is highlighted.

The BL command has no parameters.

Example

To display all the breakpoints that have been defined, use the command.

BL

- For Windows 3.1

```
0       BPMB #30:123400 W EQ 0010 DR3 C=03
1*      BPR #30:80022800 #30:80022FFF W C=01
2       BPIO 0021 W NE 00FF C=01
3       BPINT 21 AH=3D C=01
```

Note: Breakpoint 1 has an * (asterisk) following it, showing that it was disabled.

- For Windows 95 and Windows NT

```
00)     BPX #8:80102A4B IF (EAX==1) DO “DD ESI”
01)   * BPX _LockWindowInfo
02)     BPMD #013F:0063F8A0 RW DR3
03)     BPINT 2E IF (EAX==0x1E)
```

BMSG

Windows 3.1, Windows 95, Windows 98, Windows NT *Breakpoints*

Set a breakpoint on one or more Windows messages.

Syntax

For Windows 3.1

BMSG ***window-handle*** [**L**] [***begin-msg*** [***end-msg***]] [**c=*count***]

For Windows 95 and Windows NT

BMSG ***window-handle*** [**L**] [***begin-msg*** [***end-msg***]] [**IF *expression***]
[**DO "*command1;command2;...*"**]

window-handle	HWND value returned from CreateWindow or CreateWindowEX.
begin-msg	Single Windows message or lower message number in a range of Windows messages. If you do not specify a range with an end-msg, only the begin-msg will cause a break. *Note:* For both begin-msg and end-msg, the message numbers can be specified either in hexadecimal or by using the actual ASCII names of the messages, for example, WM_QUIT.
end-msg	Higher message number in a range of Windows messages.
L	Logs messages to the SoftICE Command window.
c=	Breakpoint trigger count.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, “Using Breakpoints,” in the *Using SoftICE* manual.

Use

The BMSG command is used to set breakpoints on a window's message handler that will trigger when they receive messages that either match a specified message type, or fall within an indicated range of message types.

- If you do not specify a message range, the breakpoint applies to ALL Windows messages.
- If you specify the L parameter, SoftICE logs the messages into the Command window instead of popping up when the message occurs.

When SoftICE does pop up on a BMSG breakpoint, the instruction pointer (CS:[E]IP) is on the first instruction of the message handling procedure. Each time SoftICE breaks, the current message displays in the following format:

`hWnd=`*`xxxx`* `wParam=`*`xxxx`* `lParam=`*`xxxxxxxx`* `msg=`*`xxxx`* *`message-name`*

Note: These are the parameters that are passed to the message procedure. All numbers are hexadecimal. The message-name is the Windows defined name for the message.

To display valid Windows messages, enter the WMSG command with no parameters. To obtain valid window handles, use the HWND command.

You may set multiple BMSG breakpoints on one window-handle, although the message ranges for the breakpoints may not overlap.

Example

This command sets a breakpoint on the message handler for the Window that has the handle 9BC. The breakpoint triggers and SoftICE pops up when the message handler receives messages with a type within the range WM_MOUSEFIRST to WM_MOUSELAST, inclusive (which includes all of the Windows mouse messages).

```
:BMSG 9BC wm_mousefirst wm_mouselast
```

The next command places a breakpoint on the message handler for the Window with the handle F4C. The L parameter causes the breakpoint information to be logged to the SoftICE Command window, instead of having SoftICE pop up when the breakpoint is triggered. The message range that the breakpoint triggers on includes any message with a type value less than or equal to WM_CREATE. Output from this breakpoint being triggered can be viewed by popping into SoftICE and scrolling through the command buffer.

```
:BMSG f4c L 0 wm_create
```

BPE

Windows 3.1, Windows 95, Windows 98, Windows NT *Manipulating Breakpoints*

Edit a breakpoint description.

Syntax

BPE ***breakpoint-index***

breakpoint-index	Breakpoint index number.

Use

The BPE command allows you to edit or replace an existing breakpoint. Use the editing keys to edit the breakpoint description. Press Enter to save a new breakpoint description. This command offers a quick way to modify the parameters of an existing breakpoint.

Warning: BPE first clears the breakpoint before loading it into the edit line. If you then press the Escape key, the breakpoint is cleared. To retain the original breakpoint and create another one, use the BPT command, which uses the original breakpoint as an editing template without first deleting it.

Conditional expressions and breakpoint actions are expanded as parts of the breakpoint expression.

Example

This command allows the definition for breakpoint 1 to be edited.

```
:BPE 1
```

When the command is entered, SoftICE displays the existing breakpoint definition and positions the input cursor just after the breakpoint address.

```
:BPE 1
:BPX 80104324 if (eax==1) do “dd esi”
```

To re-enter the breakpoint, press the Enter key. To clear the breakpoint, press the Escape key.

BPINT

Windows 3.1 *Breakpoints*

Set a breakpoint on an interrupt.

Syntax

BPINT *int-number* [al|ah|ax=*value*] [c=*count*]

int-number	Interrupt number from 0 - 5Fh.
value	Byte or word value.
c=	Breakpoint trigger count.

Use

Use the BPINT command to pop up SoftICE whenever a specified processor exception, hardware interrupt, or software interrupt occurs. The AX register qualifying value (AL=, AH=, or AX=) can be used to set breakpoints that trigger only when the AX register at the time that the interrupt or exception occurs matches the specified value. This capability is often used to selectively set breakpoints for DOS and BIOS calls. If an AX register value is not entered, the breakpoint occurs anytime the interrupt or exception occurs, regardless of the value of the AX register at the time.

For Windows 95 and Windows NT, refer to BPINT on page 27.

For breakpoints that trigger for hardware interrupts or processor exceptions, the instruction pointer (CS:EIP) at the time SoftICE pops up will point at the first instruction of the interrupt or exception handler routine pointed at by the IDT. If a software interrupt triggers the breakpoint, the instruction pointer (CS:EIP) points at the INT instruction that caused the breakpoint.

BPINT only works for interrupts that are handled through the IDT.

In addition, Windows maps hardware interrupts, which by default map to vectors 8-Fh and 70h-77h, to higher numbers to prevent conflicts with software interrupts. The primary interrupt controller is mapped from vector 50h-57h. The secondary interrupt controller is mapped from vector 58h-5Fh.

Example: IRQ0 is INT50h and IRQ8 is INT58h.

If a BPINT goes off due to a software interrupt instruction in a DOS VM, control will be transferred to the Windows protected mode interrupt handler for protection faults, which eventually call down to the appropriate DOS VM's interrupt handler (pointed at by the DOS VM's Interrupt Vector Table). To go directly to the DOS VM's interrupt handler after the BPINT has occurred on a software interrupt instruction, use the following command:

G @$0:*int-number4**

Example

The following command defines a breakpoint for interrupt 21h. The breakpoint occurs when DOS function call 4Ch (terminate program) is called. At the time SoftICE pops up, the instruction pointer will point at the INT instruction in the DOS VM.

```
BPINT 21 ah=4c
```

The next command sets a breakpoint that triggers on each and every tick of the hardware clock (in general this is not recommended for the obvious reason that it triggers very often!). At the time SoftICE pops up, the instruction pointer will be at the first instruction of the Windows interrupt handler for interrupt 50h.

```
BPINT 50
```

See Also

For Windows 95 and Windows NT, refer to *BPINT* on page 27.

BPINT

Windows 95, Windows 98, Windows NT *Breakpoints*

Set a breakpoint on an interrupt.

Syntax

BPINT ***int-number*** **[IF *expression*] [DO "*command1;command2;...*"]**

int-number	Interrupt number from 0 - FFh.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger
DO command	Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, "Using Breakpoints," in the *Using SoftICE* manual.

Use

Use the BPINT command to pop up SoftICE whenever a specified processor exception, hardware interrupt, or software interrupt occurs. The IF option allows arbitrary filtering of interrupts that result in breakpoints. The DO option provides the ability to associate SoftICE commands with interrupts such that they execute any time the interrupt breakpoint triggers.

For breakpoints that trigger for hardware interrupts or processor exceptions, the instruction pointer (CS:EIP) at the time SoftICE pops up will point at the first instruction of the interrupt or exception handler routine pointed at by the IDT. If a software interrupt triggers the breakpoint, the instruction pointer (CS:EIP) will point at the INT instruction that caused the breakpoint.

BPINT only works for interrupts that are handled through the IDT.

For Windows 3.1, refer to BPINT on page 25.

If a software interrupt occurs in a DOS VM, control is transferred to a Windows protected mode interrupt handler, which eventually calls down to the DOS VM's interrupt handler (pointed at by the DOS VM's Interrupt Vector Table). To go directly to the DOS VM's interrupt handler after the BPINT has occurred on a software interrupt instruction, use the following command:

G @ &0:(*int-number4)**

For Windows 95

Windows maps hardware interrupts, which by default map to vectors 8-Fh and 70h-77h, to higher numbers to prevent conflicts with software interrupts. The primary interrupt controller is mapped from vector 50h-57h. The secondary interrupt controller is mapped from vector 58h-5Fh.

Example: IRQ0 is INT50h and IRQ8 is INT58h.

For Windows NT

Windows NT maps hardware interrupts, which by default map to vectors 8-Fh and 70h-77h, to higher numbers to prevent conflicts with software interrupts. The primary interrupt controller is mapped from vector 30h-37h. The secondary interrupt controller is mapped from vector 38h-3Fh.

Example: IRQ0 is INT30h and IRQ8 is INT38h

Example

The following example results in Windows NT system call (software interrupt 2Eh) breakpoints only being triggered if the thread making the system call has a thread ID (TID) equal to the current thread at the time the command is entered (_TID). Each time the breakpoint hits, the contents of the address 82345829h are dumped as a result of the DO option.

```
BPINT 2e if tid==_tid do "dd 82345829"
```

See Also

For Windows 3.1, refer to *BPINT* on page 25.

BPIO

Windows 3.1, Windows 95, Windows 98, Windows NT *Breakpoints*

Set a breakpoint on an I/O port access.

Syntax

For Windows 3.1

BPIO ***port*** [***verb***] [***qualifier value***] [**c=*count***]

For Windows 95

BPIO [-h] ***port*** [***verb***] [**IF *expression***] [**DO "*command1;command2;...*"**]

For Windows NT

BPIO ***port*** [***verb***] [**IF *expression***] [**DO "*command1;command2;...*"**]

port Byte or word value.

verb

Value	Description
R	Read (IN)
W	Write (OUT)
RW	Reads and Writes

qualifier

Qualifier, value, and C= are not valid for Windows 95 and Windows NT.

Value	Description
EQ	Equal
NE	Not Equal
GT	Greater Than
LT	Less Than
M	Mask. A bit mask is represented as a combination of 1's, 0's and X's. X's are don't-care bits.

value Byte, word, or dword value.

c= Breakpoint trigger count.

-h	Use hardware debug registers to set a breakpoint in Vxd. Available for Pentium-class processors on Windows 95 only.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, "Using Breakpoints," in the *Using SoftICE* manual.

Use

Use the BPIO instruction to have SoftICE pop up whenever a specified I/O port is accessed in the indicated manner. When a BPIO breakpoint triggers, the instruction pointer (CS:EIP) points to the instruction following the IN or OUT instruction that caused the breakpoint.

If you do not specify a verb, RW is the default.

For Windows 3.1

If you specify verb and value parameters, the value specified is compared with, according to the verb, the actual data value read or written by the IN or OUT instruction causing the breakpoint. The value may be a byte, a word, or a dword. The possible verbs allow for comparisons of equality, inequality, greater-than-or-equal, less-than-or-equal, and logical AND comparison.

For Windows 3.1 and Windows 95

Due to the behavior of the x86 architecture, BPIO breakpoints are only active while the processor is executing in the RING 3 privilege level. This means that I/O activity performed by RING 0 code such as VxDs and the Windows VMM are not trapped by BPIO breakpoints. For Windows 95 only, use the -H switch to force SoftICE to use the hardware debug registers. This lets you trap I/O performed at Ring 0 in VxDs.

Windows virtualizes many of the system I/O ports, meaning that VxDs have registered handlers that are called when RING 3 accesses are made to the ports. To get a list of virtualized ports, use the TSS command. The command shows each hooked I/O port plus the address of its associated handler and the name of the VxD that owns it. To see how a particular port is virtualized, set a BPX on the address of the I/O handler.

For Windows NT

The BPIO command uses the debug register support provided on the Pentium, therefore, I/O breakpoints are only available on Pentium-class machines.

When using debug registers for I/O breakpoints, all physical I/O instructions (non-emulated) are trapped no matter what privilege level they are executed from. This is different from using the I/O bit map to trap I/O, as is done for SoftICE running under Windows 3.1 and Windows 95 (without the -H switch). The I/O bit map method can only trap I/O done from user-level code, whereas a drawback of the debug register method for trapping port I/O is that it does not trap emulated I/O such as I/O performed from a DOS box.

Due to limitations in the number of debug registers available on x86 processors, a maximum of four BPIOs can be set at any given time.

Example

The following commands define conditional breakpoints for accesses to port 21h (interrupt control 1's mask register). The breakpoints only trigger if the access is a write access, and the value being written is not FFh.

- For Windows 3.1

  Use this command: **`BPIO 21 w ne ff`**

- For Windows 95 and Windows NT

  Use this command: **`BPIO 21 w if (al!=0xFF)`**

  Note: In the Windows NT example, you should be careful about intrinsic assumptions being made about the size of the I/O operations being trapped. The port I/O to be trapped is OUTB. An OUTW with AL==FFh also triggers the breakpoint, even though in that case the value in AL ends up being written to port 22h.

The following example defines a conditional byte breakpoint on reads of port 3FEh. The breakpoint occurs the first time that I/O port 3FEh is read with a value that has the two high-order bits set to 1. The other bits can be of any value.

- For Windows 3.1

  Use this command: **`BPIO 3fe r eq m 11xx xxxx`**

- For Windows 95 and Windows NT

  Use this command: **`BPIO 3fe r if ((al & 0xC0)==0xC0)`**

BPM

Windows 3.1, Windows 95, Windows 98, Windows NT *Breakpoints*

Set a breakpoint on memory access or execution.

Syntax

For Windows 3.1

BPM[***size***] ***address*** [***verb***] [***qualifier value***] [***debug-reg***] [**c=*count***]

For Windows 95 and Windows NT

BPM[***size***] ***address*** [***verb***] [***debug-reg***] [**IF *expression***]
[**DO "*command1;command2;...*"**]

size

Size is actually a range covered by this breakpoint. For example, if you use double word, and the third byte of the dword is modified, a breakpoint occurs. The size is also important if you specify the optional qualifier.

Value	Description
B	Byte
W	Word
D	Double Word

verb

Value	Description
R	Read
W	Write
RW	Reads and Writes
X	Execute

Qualifier, value, and C= are not valid for Windows 95 and Windows NT.

qualifier	These qualifiers are only applicable to read and write breakpoints, not execution breakpoints.

Value	Description
EQ	Equal
NE	Not Equal
GT	Greater Than
LT	Less Than
M	Mask. A bit mask is represented as a combination of 1's, 0's and X's. The X's are don't-care bits.

value	Byte, word, or double word value, depending on the size you specify.
debug-reg	

Value
DR0
DR1
DR2
DR3

c=	Breakpoint trigger count.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, “Using Breakpoints,” in the *Using SoftICE* manual.

Use

Use BPM breakpoints to have SoftICE pop up whenever certain types of accesses are made to memory locations. The size and verb parameters allow for the accesses to be filtered according to their type, and the DO parameter (Windows NT only) allows for arbitrary SoftICE commands to be executed each time the breakpoint is hit.

If you do not specify a debug register, SoftICE uses the first available debug register starting from DR3 and working backwards. You should not include a debug register unless you are debugging an application that uses debug registers itself such as a debugging tool.

If you do not specify a verb, RW is the default.

If you do not specify a size, B is the default.

For all the verb types except X, SoftICE pops up *after* the instruction that causes the breakpoint to trigger has executed. The CS:EIP points at the instruction in the code stream following the trapped instruction. In the case of the X verb, SoftICE pops up *before* the instruction causing the breakpoint to trigger has executed. The CS:EIP therefore points at the instruction where the breakpoint was set.

If you specify the R verb, breakpoints occur on read accesses and on write operations that do not change the value of the memory location.

If the verb is R, W or RW, *executing* an instruction at the specified address does not cause the breakpoint to occur.

If you set a breakpoint using BPMW it is a word-sized memory breakpoint, then the specified address must start on a word boundary. If you set a breakpoint using BPMD the memory breakpoint is dword sized, then the specified address must start on a double word boundary.

For Windows 3.1

The count parameter can be used to have a breakpoint trigger only after it has been hit a specified number of times. The default count value is 1, meaning that the breakpoint triggers the first time the breakpoint condition is satisfied. The count is reset each time the breakpoint triggers.

For Windows 95

BPM breakpoints set in the range 400000 - 7FFFFFFF (WIN32 applications) are address-context sensitive. That is, they are triggered only when the address context in which the breakpoint was set is active. If a BPM is set in a DLL that exists in multiple contexts, the breakpoint is armed in all the contexts in which it exists. For example, if you set a BPM X breakpoint in KERNEL32 it could break in any context that contains KERNEL32.DLL.

For Windows NT

Any breakpoint set on an address below 80000000h (2 GB) is address-context sensitive. This includes WIN32 and DOS V86 applications. Take care to ensure you are in the correct context before setting a breakpoint.

Example

The following example defines a breakpoint on memory byte access to the address pointed at by ES:DI+1Fh. The first time that 10h is written to that location, the breakpoint triggers.

- For Windows 3.1

  Use the command: **`BPM es:di+1f w eq 10`**

- For Windows 95 and Windows NT

  Use the command: **`BPM es:di+1f w if (*(es:di+1f)==0x10)`**

The next example defines an execution breakpoint on the instruction at address CS:80204D20h. The first time that the instruction at the address is executed, the breakpoint occurs.

- For Windows 3.1, Window 95, and Windows NT

  Use the command: **`BPM CS:80204D20 x`**

The following example defines a word breakpoint on a memory write. The breakpoint occurs the first time that location `Foo` has a value written to it that sets the high order bit to 0 and the low order bit to 1. The other bits can be any value.

- For Windows 3.1

  Use the command: **`BPMW foo e eq m 0xxx xxxx xxxx xxx1`**

This example sets a byte breakpoint on a memory write. The breakpoint triggers the first time that the byte at location DS:80150000h has a value written to it that is greater than 5.

- For Windows 3.1

  Use the command: **`BPM ds:80150000 w gt 5`**

- For Windows 95 and Windows NT

  Use the command: **`BPM ds:80150000 if (byte(*ds:80150000)>5)`**

BPR

Windows 3.1, Windows 95, Windows 98 *Breakpoints*

Set a breakpoint on a memory range.

Syntax

For Windows 3.1

BPR ***start-address end-address*** [***verb***] [**c=*count***]

For Windows 95

BPR ***start-address end-address*** [***verb***] [**IF** ***expression***]
[**DO "*command1;command2;...*"**]

start-address Beginning of memory range.

end-address Ending of memory range.

verb

Value	Description
R	Read
W	Write
RW	Reads and Writes
T	Back Trace on Execution
TW	Back Trace on Memory Writes

c= Breakpoint trigger count.

IF expression Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.

DO command Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, “Using Breakpoints,” in the *Using SoftICE* manual.

Use

Use the BPR command to set breakpoints that trigger whenever certain types of accesses are made to an entire address range.

There is no explicit range breakpoint for execution access, however, execution breakpoints on ranges can be obtained with the R verb. An instruction fetch is considered a read for range breakpoints.

If you do not specify a verb, W is the default.

The range breakpoint degrades system performance in certain circumstances. Any read or write within the 4KB page that contains a breakpoint range is analyzed by SoftICE to determine if it satisfies the breakpoint condition. This performance degradation is usually not noticeable, however, degradation could be extreme in cases where there are frequent accesses to the range.

The T and TW verbs enable back trace ranges on the specified range. They do not cause breakpoints, but instead result in information about all instructions that would have caused the breakpoint to trigger to be written to a log that can be displayed with the SHOW or TRACE commands.

When a range breakpoint is triggered and SoftICE pops up, the current CS:EIP points at the instruction that caused the breakpoint.

Range breakpoints are always set in the page tables that are active when the BPR command is entered. Therefore, if range addresses are below 4MB, the range breakpoint will be tied to the virtual machine that is current when BPR is entered. Because of this fact, there are some areas in memory where range breakpoints are not supported. These include the page tables, GDT, IDTs, LDT, and SoftICE. If you try to set a range breakpoint or back trace range over one of these areas, SoftICE returns an error.

There are two other data areas in which you cannot place a range breakpoint, but if you do SoftICE will not complain. These are Windows level 0 stacks and critical areas in the VMM. Windows level 0 stacks are usually in separately allocated data segments. If you set a range over a level 0 stack or a critical area in VMM, you could hang the system.

If the memory that covers the range breakpoint is swapped or moved, the range breakpoint follows it.

For Windows 3.1

The count parameter can be used to have a breakpoint trigger only after it has been hit a specified number of times. The default count value is 1, meaning that the breakpoint will trigger the first time the breakpoint condition is satisfied. The count is reset each time the breakpoint triggers.

For Windows 95

Due to a change in system architecture, BPRs are no longer supported in level 0 code. Thus, you cannot use BPRs to trap VxD code.

Example

The following example defines a breakpoint on a memory range. The breakpoint occurs if there are any writes to the memory between addresses ES:0 and ES:1FFF:

```
BPR es:0 es:1fff w
```

BPRW

Windows 3.1, Windows 95, Windows 98 *Breakpoints*

Set range breakpoints on Windows program or code segment.

Syntax

For Windows 3.1

BPRW *module-name* | *selector* [*verb*]

For Windows 95

BPRW *module-name* | *selector* [*verb*] [IF *expression*]
[DO "*command1;command2;...*"]

module-name	Any valid Windows Module name that contains executable code segments.
selector	Valid 16-bit selector in a Windows program.
verb	

Value	Description
R	Read
W	Write
RW	Reads and Writes
T	Back Trace on Execution
TW	Back Trace on Memory Writes

IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, “Using Breakpoints,” in the *Using SoftICE* manual.

Use

The BPRW command is a short-hand way of setting range breakpoints on either all of the code segments, or on a single segment of a Windows program.

The BPRW command actually sets BPR style breakpoints. Thus, if you enter the BL command after entering a BPRW command, you can see where separate range breakpoints were set to cover the segments specified in the BPRW command.

Valid selectors for a 16-bit Windows program can be obtained with the HEAP instruction.

Clearing the breakpoints created by BPRW commands requires that each of these range breakpoints be separately cleared with the BC command.

Note: The BPRW command can become very slow when using the T verb to back trace or when using the command in conjunction with a CSIP qualifying range.

For Windows 95

Due to a change in system architecture, BPRs are no longer supported in level 0 code. For example, you cannot use BPRs to trap VxD code.

When a BPRW is set on a 32-bit application or DLL, a single range breakpoint is set starting at the executable image base and ending at the image base plus image size.

Common Uses

The BPRW command is commonly used to do the following:

- To set a back trace history range over an entire Windows application or DLL, specify the module-name and the T verb.
- To set a breakpoint that triggers whenever a program executes, use the R verb. This works because the R verb breaks on execution as well as reads.
- To use BPRW as a convenient form of BPR. Instead of requiring you to look up a segment's base and limit through the LDT or GDT commands, you only need to know the segment selector.

Example

This example sets up a back trace range on all of the code segments in the Program Manager. All instructions that the Program Manager executes are logged to the back trace history buffer and can later be viewed with the TRACE and SHOW commands.

```
BPRW progman t
```

BPT

Windows 3.1, Windows 95 , Windows 98 *Manipulating Breakpoints*

Use a breakpoint description as a template.

Syntax

BPT *breakpoint-index*

breakpoint-index Breakpoint index number.

Use

The BPT command uses an existing breakpoint description as a template for defining a new breakpoint. The BPT command loads a template of the breakpoint description into the edit line for modification. Use the editing keys to edit the breakpoint description and type Enter to add the new breakpoint description. The breakpoint referenced by breakpoint index is not altered. This command offers a quick way to modify the parameters of an existing breakpoint.

Conditional expressions are expanded as parts of the breakpoint expression as well as breakpoint actions.

Example

The following example moves a template of breakpoint 3 into the edit line (without removing breakpoint 3). An example of the edit line follows:

```
BPT 3
:BPX 1b:401200 if (eax==1) do “dd esi”
```

Press Enter to add the new breakpoint.

BPX

Windows 3.1, Windows 95, Windows 98, Windows NT *Breakpoints*

F9

Set or clear a breakpoint on execution.

Syntax

For Windows 3.1

BPX [***address***] [**c=*count***]

For Windows 95 and Windows NT

BPX [***address***] [**IF *expression***] [**DO "*command1;command2;...*"**]

address	Linear address to set execution breakpoint.
c=	Breakpoint trigger count.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of SoftICE commands can execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, BPLOG, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered. See Chapter 6, “Using Breakpoints,” in the *Using SoftICE* manual.

Use

Use the BPX command to define breakpoints that trigger whenever the instruction at the specified address is executed.

The address parameter must point at the first byte of the instruction opcode of the instruction where the breakpoint is being set. If no address is specified and the cursor is in the Code window when you begin to type the command, a point-and-shoot breakpoint is set where the implied address is that of the instruction at the cursor location in the Code window. If you define a point-and-shoot breakpoint at an address where a breakpoint already exists, the existing breakpoint is cleared.

Note: Use the EC command (default key F6) to move the cursor into the Code window.

If the cursor is not in the Code window when you enter the BPX command, you must specify an address. If you specify only an offset, the current CS register value is used as the segment.

The BPX command normally places an INT 3 instruction at the breakpoint address. This breakpoint method is used instead of assigning a debug register to make more execution breakpoints available. If you need to use a breakpoint register, for example, to set a breakpoint on code not yet loaded in a DOS VM, set an execution breakpoint with the BPM command and specify X as the verb.

If you try to set a BPX at an address that is in ROM, a breakpoint register is automatically used for the breakpoint instead of the normal placement of an INT 3 at the target address (because ROM cannot be modified).

The BPX command accepts 16-bit Windows module names as an address parameter. When you enter a 16-bit module name, SoftICE sets a BPX-style breakpoint on every exported entry point in the module.

Example: BPX KERNEL sets a breakpoint on every function in the 16-bit Windows module KRNL386.EXE. This can be very useful is you need to break the next time any function in a DLL is called.

SoftICE supports a maximum of 256 breakpoints when using this command.

For Windows 3.1 and Windows 95

BPX breakpoints in DOS VMs are tied to the VM they were set in. This is normally what you would like when debugging a DOS program in a DOS VM. However, there are situations when you may want the breakpoint to go off at a certain address no matter what VM is currently mapped in. This is usually true when debugging in DOS code or in a TSR that was run before Windows was started. In these cases, use a BPM breakpoint with the X verb instead of BPX.

For Windows 95

BPX breakpoints set in the range 400000 - 7FFFFFFF (WIN32 applications) are address-context sensitive. That is, they are only triggered when the context in which they were set is active. If a breakpoint is set in a DLL that exists in multiple contexts, however, the breakpoint will exist in all contexts.

For Windows NT

Any breakpoint set on an address below 80000000h (2 GB) is address-context sensitive. This includes WIN32, WIN16, and DOS V86 applications. Take care to ensure you are in the correct context before setting a breakpoint.

Example

This example sets an execution breakpoint at the instruction 10h bytes past the current instruction pointer (CS:EIP).

```
BPX eip+10
```

This example sets an execution breakpoint at source line 1234 in the current source file (refer to *FILE* on page 81).

```
BPX .1234
```

For Windows 95 and Windows NT

The following is an example of the use of a conditional expression to qualify a breakpoint. In this case, the breakpoint triggers if the EAX register is within the specified range:

```
BPX eip if eax > 1ff && eax <= 300
```

In this example, a breakpoint action is used to have SoftICE automatically dump a parameter for a call. Every time the breakpoint is hit, the contents of the string pointed to by the current DS:DX will be displayed in the Data window.

```
BPX 80023455 do “db ds:dx”
```

See Also

FILE

BSTAT

Windows 95, Windows 98, Windows NT *Breakpoints*

Display statistics for one or more breakpoints.

Syntax

BSTAT [***breakpoint-index***]

breakpoint-index	Breakpoint index number.

Use

Use BSTAT to display statistics on breakpoint hits, misses, and whether breakpoints popped up or were logged. A breakpoint will be logged to the history buffer instead of popping up if it has a conditional expression that uses the BPLOG expression macro.

Because conditional expressions are evaluated when the breakpoint is triggered, it is possible to have evaluation run-time errors. Examples of this are when a virtual symbol is referenced, and that symbol has not been loaded, or a reference to symbol cannot be resolved because the memory is not present. In these cases, and possibly others, an error will be generated and noted. The Status and Scode fields under the Misc. column contain error information which indicates what problem, if any, has occurred.

Output

For each breakpoint displayed the following information also appears:

BP #	Breakpoint index, and if disabled, an * (asterisk).
Totals Category:	
Hits	Total number of times SoftICE has evaluated the breakpoint.
Breaks	Total number of times the breakpoint has evaluated TRUE, and SoftICE has either popped up, or logged the breakpoint.
Popups	Total number of times the breakpoint caused SoftICE to pop up.
Logged	Total number of times the breakpoint has been logged.
Misses	Total number of times the breakpoint evaluated to FALSE, and no breakpoint action was taken.
Errors	Total number of times that the evaluation of a breakpoint resulted in a error.
Current Category:	
Hits	Current number of times the breakpoint has evaluated TRUE, but did not pop up because the count had not expired. (Refer to expression macro BPCOUNT.)
Misses	Current number of times the breakpoint has evaluated FALSE and/or

the breakpoint count has not expired.

Miscellaneous Category:

Status	SoftICE internal status code for the last time the breakpoint was evaluated, or zero if no error occurred.
Scode	Last non-zero SoftICE internal status code, or zero if no error has occurred.
Cond.	Yes if the breakpoint has a conditional expression, otherwise No.
Action	Yes if the breakpoint has a defined breakpoint action, otherwise No.

Example

The following is an example using the BSTAT command for breakpoint #0:

```
:BSTAT 0

Breakpoint Statistics for #00
    BP #       *00

Totals
    Hits       2
    Breaks     2
    Popups     2
    Logged     0
    Misses     0
    Errors     0

Current
    Hits       0
    Misses     0

Misc
    Status     0
    SCode      0
    Cond.      No
    Action     Yes
```

See Also

For more information on breakpoint evaluation, refer to *Using SoftICE.*

C

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Compare two data blocks.

Syntax

C *start-address* l *length* *start-address-2*

start-address	Start of first memory range.
length	Length in bytes.
start-address-2	Start of second memory range.

Use

The memory block specified by start-address and length is compared to the memory block specified by the second start address.

When a byte from the first data block does not match a byte from the second data block, both bytes display, along with their addresses.

Example

The following example compares 10h bytes starting at memory location DS:805FF000h to the 10h bytes starting at memory location DS:806FF000h.

```
C ds:805ff000 l 10 ds:806ff000
```

CLASS

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display information on Window classes.

Syntax

For Windows 3.1

CLASS [*module-name*]

For Windows 95

CLASS [-x][*task-name*]

For Windows NT

CLASS [-x][*process-type* | *thread-type* | *module-type* | *class-name*]

module-name	Any currently loaded Windows module. Not all Windows modules have classes registered.
-x	Display complete Windows 95 or Windows NT internal CLASS data structure, expanding appropriate fields into more meaningful forms.
task-name	Any currently executing 16- or 32-bit task.
process-type	Process name, process ID, or process handle.
thread-type	Thread ID or thread address (KTEB).
module-type	Module name or module handle.
class-name	Name of a registered class window.

Use

For Windows 95

The operating system maintains the standard window classes in the 16-bit user module (per Windows 3.1). The operating system maintains all other window classes in separate lists on behalf of each process. Each time a process or one of its DLLs registers a new window class, registration places that class on one of two lists:

- The application global list contains classes registered with the CS_GLOBAL attribute. They are accessible to the process or any of its DLLs.
- The application private list contains non-global classes. Only the registering module can access them.

Finally, any process or DLL that attempts to superclass one of the standard window controls, for example, LISTBOX, receives a copy of that class. The copy resides in a process-specific system-superclass list. By making a copy of the standard class, a process or DLL can superclass any standard windows control without affecting other processes in the system.

The process-specific class lists display in the following order:

- application private
- application global
- system superclassed

In the output, dashed lines separate each list.

For Windows NT

The architecture of class information under Windows NT is similar to that of Windows 95 in that class information is process specific and the operating system creates different lists for global and private classes. Beyond this, the two operating systems have significant differences in how super-classing a registered window class is implemented.

Under Windows NT, registered window classes are considered *templates* that describe the base characteristics and functionality of a window (similar to the C++ notion of an abstract class). When a window of any class is created, the class template is *instanced* by making a physical copy of the class structure. This instanced class is stored with the windows instance data. Any changes to the instanced class data does not affect the original class template. This concept is further extended when various members of the windows instanced class structure are modified. When this occurs, the instanced class is instanced again, and the new instance points to the original instance. Registered classes act as templates from which instances of a particular class can be created; in effect this is object inheritance. This inheritance continues as changes are made to the base functionality of the class.

If you do not specify the type parameter, the current context is assumed because the class information is process specific. A process-name always overrides a module of the same name. To search by module when there is a name conflict, use the module handle (base address or module database selector). Also, module names are *always* context sensitive. If the module is not loaded in the current context (or the CSRSS context), the CLASS command interprets the module name as a class name instead.

Output

For each class, the following information is shown:

Class Handle	Offset of a data structure within USER. Refers to windows of this class.
Class Name	Name that was passed when the class was registered. If no name was passed, the atom displays.
Owner	Module that has registered this window class.

Window Procedure	Address of the window procedure for this window class.
Styles	Bitmask of flags specified when the class was registered.

Example

For Windows 3.1

The following example uses the CLASS command to display all the classes registers by the MSWORD module.

:CLASS msword

```
Handle     Name          Owner      Window Procedure
0F24        #32772       USER        TITLEWNDPROC
0EFC        #32771       USER        SWITCHWNDPROC
0ED4        #32769       USER        DESKTOPWNDPROC
0E18        MDIClient    USER        MDICLNTWNDPROC
0DDC        ComboBox     USER        COMBOBXWNDPROC
0DA0        ComboLBox    USER        LBBOXTLWNDPROC
0D64        ScrollBar    USER        SBWNDPROC
0D28        ListBox      USER        LBOXCTLWNDPROC
0CF0        Edit         USER        EDITWNDPROC
```

Note: There are symbols for all of the window procedures, because SoftICE includes all of the exported symbols from USER.EXE. If a symbol is not available for the window procedure, a hexadecimal address displays.

CLS

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Alt-F5

Clear the Command window.

Syntax

CLS

Use

The CLS command clears the SoftICE Command window, all display history, and moves the prompt and the cursor to the upper lefthand corner of the Command window.

CODE

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Display instruction bytes.

Syntax

`CODE [on | off]`

Use

The CODE command controls whether or not the actual hexadecimal bytes of an instruction display when the instruction is unassembled.

- If CODE is ON, the instruction bytes display.
- If CODE is OFF, the instruction bytes do not display.
- CODE with no parameters displays the current state of CODE.
- The default is CODE mode OFF.

Example

The following command causes the actual hexadecimal bytes of an instruction to display when the instruction is unassembled.

```
CODE on
```

See Also

SET

COLOR

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Display or set the screen colors.

Syntax

`COLOR [normal bold reverse help line]`

normal	Foreground/background attribute that displays normal text. Default = 07h grey on black.
bold	Foreground/background attribute that displays bold text. Default = 0Fh white on black.
reverse	Foreground/background attribute that displays reverse video text. Default = 71h blue on grey.
help	Foreground/background attribute that displays the help line underneath the Command window. Default = 30h black on cyan.
line	Foreground/background attribute that displays the horizontal lines between the SoftICE windows. Default = 02h green on black.

Use

Use the COLOR command to customize the SoftICE screen colors on a color monitor. Each of the five specified colors is a hexadecimal byte where the foreground color is in bits 0-3 and the background color is in bits 4-6. This is identical to the standard CGA attribute format where there are 16 foreground colors and 8 background colors.

The actual colors represented by the 16 possible codes are listed in the following table:

Code	Color	Code	Color
0	black	A	light green
1	blue	B	light cyan
2	green	C	light red
3	cyan	D	light magenta
4	red	E	yellow
5	magenta	F	white

Code	Color	Code	Color
6	brown		
7	grey		
8	dark grey		
9	light blue		

Example

This command causes the following color assignments:

COLOR 7 f 71 30 2

normal text	grey on black
bold text	white on black
reverse video text	blue on grey
help line	black on cyan
horizontal line	green on black

CPU

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the registers.

Syntax

CPU [-i]

-i	Displays the I/O APIC.

Use

The CPU command shows all the CPU registers (general, control, debug, and segment).

For Windows NT

If your PC contains a multi-processor mother board that uses an I/O APIC as an interrupt controller, the CPU command displays the CPU local and I/O APICS.

Example

The following example lists the sample output from the CPU command under Windows 95 or Windows NT on systems that do not use an I/O APIC:

```
Processor 00 Registers
----------------------
CS:EIP=0008:8013D7AE   SS:ESP=0010:8014AB7C
EAX=00000041   EBX=FFDFF000   ECX=00000041   EDX=80010031
ESI=80147940   EDI=80147740   EBP=FFDFF600   EFL=00000246
DS=0023   ES=0023   FS=0030   GS=0000

CR0=8000003F PE MP EM TS ET NE PG
CR2=C13401D6
CR3=00030000
CR4=00000011 VME PSE
DR0=00000000
DR1=00000000
DR2=00000000
DR3=00000000
DR6=FFFF0FF0
DR7=00000400
EFL=00000246 PF ZF IF IOPL=0
```

The following example lists the sample output from the CPU command under Windows NT on a system that uses an I/O APIC:

```
Processor 00 Registers
----------------------
CS:EIP=0008:8013D7AE   SS:ESP=0010:8014AB7C
EAX=00000041   EBX=FFDFF000   ECX=00000041   EDX=80010031
ESI=80147940   EDI=80147740   EBP=FFDFF600   EFL=00000246
DS=0023   ES=0023   FS=0030   GS=0000

CR0=8000003F PE MP EM TS ET NE PG
CR2=C13401D6
CR3=00030000
CR4=00000011 VME PSE
DR0=00000000
DR1=00000000
DR2=00000000
DR3=00000000
DR6=FFFF0FF0
DR7=00000400
EFL=00000246 PF ZF IF IOPL=0

--------Local apic--------
                   ID: 0
              Version: 30010
        Task Priority: 41
 Arbitration Priority: 41
   Processor Priority: 41
   Destination Format: FFFFFFFF
  Logical Destination: 1000000
      Spurious Vector: 11F
    Interrupt Command: 3000000:60041
          LVT (Timer): 300FD
          LVT (Lint0): 1001F
          LVT (Lint1): 84FF
          LVT (Error): E3
          Timer Count: 3F94DB0
        Timer Current: 23757E0
         Timer Divide: B
```

The following example lists the sample output from the CPU -i command under Windows NT on a system that uses an I/O APIC:

```
Inti   Vector   Delivery   Status   Trigger   Dest Mode   Destination
01     91       Low. Pri   Idle     Edge      Logical     01000000
03     61       Low. Pri   Idle     Edge      Logical     01000000
04     71       Low. Pri   Idle     Edge      Logical     01000000
08     D1       Fixed      Idle     Edge      Logical     01000000
0C     81       Low. Pri   Idle     Edge      Logical     01000000
0E     B1       Low. Pri   Idle     Edge      Logical     01000000
I/O unit id register: 0E000000
I/O unit version register: 000F0011
```

See Also

PAGE

CR

Windows 3.1 *System Information*

Display the control registers.

Syntax

CR

Use

The CR command displays the contents of the three control registers CR0, CR2, and CR3, and the debug registers in the Command window. CR0 is the processor control register. CR2 is the register in which the processor stores the most recently accessed address that resulted in a page fault. CR3 contains the *physical* address of the system's page directory (refer to *PAGE* on page 150).

Example

The following example lists the sample output from a CR command:

```
CR0=8000003B PE MP TS ET NE PG
CR2=000CC985
CR3=002FE000
CR4=00000008 DE
DR1=00000000
DR2=00000000
DR3=00000000
DR6=FFFF0FF0
DR7=00000400
```

See Also

PAGE

CSIP

Windows 3.1 *Breakpoints*

Set CS:EIP (instruction pointer) memory range qualifier for all breakpoints (for 16-bit programs only).

Syntax

CSIP [off | [not] ***start-address end-address*** **|** ***Windows-module-name*****]**

off	Turns off CSIP checking.
not	Breakpoint only occurs if the CS:EIP is outside the specified range.
start-address	Beginning of memory range.
end-address	End of memory range.
Windows-module-name	If you specify a valid Windows-module-name instead of a memory range, the range covers all code areas in the specified Windows module.

Use

For Windows 3.1

The CSIP command qualifies breakpoints so that the code that causes the breakpoint must come from a specified memory range. This function is useful when a program is suspected of accidentally modifying memory outside of its boundaries.

When breakpoint conditions are met, the instruction pointer (CS:EIP) is compared to the specified memory range. If it is within the range, the breakpoint activates. To activate the breakpoint only when the instruction pointer (CS:EIP) is outside the range, use the NOT parameter.

Because 16-bit Windows programs are typically broken into several code segments scattered throughout memory, you can input a Windows module name as the range. If you enter a module name, the range covers all code segments in the specified Windows program or DLL.

When you specify a CSIP range, it applies to ALL breakpoints that are currently active.

If do not specify parameters, the current memory range displays.

For Windows 95 and Windows NT

For 32-bit code, this command is obsolete. Use conditional expressions to achieve this functionality. CSIP still works for 16-bit code and modules.

Example

The following command causes breakpoints to occur only if the CS:EIP is NOT in the ROM BIOS when the breakpoint conditions are met.

```
CSIP not $f000:0 $ffff:0
```

The following command causes breakpoints to occur only if the Windows program CALC causes them.

```
CSIP calc
```

D

Windows 3.1, Windows 95, Windows 98, Windows NT *Display/Change Memory*

Display memory.

Syntax

For Windows 3.1

D[***size***] [***address***]

For Windows 95 and Windows NT

D[***size***] [***address*** [**l** ***length***]]

size

Value	Description
B	Byte
W	Word
D	Double Word
S	Short Real
L	Long Real
T	10-Byte Real

Use

The D command displays the memory contents at the specified address.

The contents display in the format of the size you specify. If you do not specify a size, the last size used displays. The ASCII representation displays for the byte, word, and double word hexadecimal formats.

For the dword format, data is displayed in two different ways.

- If the displayed segment is a 32-bit segment, the dwords display as 32-bit hexadecimals (eight hexadecimal digits).
- If the displayed segment is a 16-bit segment (VM segment or LDT selector), the dwords display as 16:16 pointers (four hexadecimal digits ':' four more hexadecimal digits).

If you do not specify an address, the command displays memory at the next sequential address after the last byte displayed in the current Data window.

If the Data window is visible, the data displays there; otherwise, it displays in the Command window. In the Command window, either eight lines display or one less than the length of the window.

For floating point values, numbers can display in the following format:

```
[leading sign] decimal-digits . decimal-digits E sign exponent
```

The following ASCII strings can also display for real formats:

String	Exponent	Mantissa	Sign
Not A Number	all 1's	NOT 0	+/-
Denormal	all 0's	NOT 0	+/-
Invalid	10 byte only with mantissa=0		
Infinity	all 1's	0	+/-

For Windows 95 and Windows NT

If an L parameter followed by a length is specified, SoftICE displays the requested number of bytes to the Command window regardless of whether the Data window is visible. SoftICE always displays whole rows. If the length is not a multiple of rows, SoftICE will round up. This command is useful when dumping large amounts of data to the Command window for the purpose of logging it to a file.

Example

Displays the memory starting at address ES:1000h in word format and in ASCII format.

```
DW es:1000
```

For Windows 95 and Windows NT

The following command displays 4KB of memory starting at address SS:ESP in dword format. The data is displayed in the Command window.

```
:DD ss:esp l 1000
```

DATA

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Windows 3.1 - F12

Change to display another Data window.

Syntax

DATA [*window-number*]

window-number	Number of the Data window you want to view. This can be 0, 1, 2, or 3.

Use

SoftICE supports up to four Data windows. Each Data window can display a different address and/or format. Only one Data window is visible at any time. Specifying DATA without a parameter just switches to the next Data window. The windows are numbered from 0 to 3. This number displays on the righthand side of the line above the Data window. If you specify a window-number after the DATA command, SoftICE switches to display that window. The DATA command is most useful when assigned to a function key. See Chapter 8, “Customizing SoftICE,” in the *Using SoftICE* manual.

Example

Changes the Data window to Data window number 3.

```
DATA 3
```

DEVICE

Windows 98, Windows NT *System Information*

Display information on Windows NT devices.

Syntax

DEVICE [***device-name*** | ***pdevice-object***]

Use

The DEVICE command displays information on Windows NT device objects. If the DEVICE command is entered without parameters, summary information displays for all device objects found in the \Device directory. However, if a specific device object is indicated, either by its object directory name (device-name) or object address (pdevice-object), more detailed information displays.

If a directory is not specified with a device-name, the DEVICE command attempts to locate the named device object in the \Device object directory. To display information about a device object that is not located in the \Device directory, specify the complete object path name of the device object. When displaying information about a specified device, the DEVICE command displays fields of the DEVICE_OBJECT data structure as defined in NTDDK.H.

Output

The following fields are shown as summary information:

RefCnt	Device object's reference count.
DrvObj	Pointer to the driver object that owns the device object.
NextDev	Pointer to the next device object on the linked list of device objects that were created by the same driver.
AttDev	Pointer to a device object that has been attached to the displayed object via an IoAttachDeviceObject call. Attached device objects are essentially IRP filters for the devices to which they are attached.
CurIrp	Pointer to the IRP currently being serviced for the device object by the device object's driver.
DevExten	Pointer to device driver-defined device object extension data structure.
Name	Name of the device, if it has one.

The following are some fields shown when detailed information is printed:

Flags	Definition of the device object's attributes such as whether I/O performed on the device is buffered or not.
Vpb	Pointer to the device's associated volume parameter block.
Device Type	User-defined or pre-defined value that SoftICE translates to a name.

Example

The following example shows the DEVICE command output with no parameters. It results in SoftICE printing summary information on all device objects in the \Device object directory.

DEVICE

RefCnt	DrvObj	NextDev	AttDev	CurIrp	DevExten	Name
00000000	FD8CD910	00000000	00000000	00000000	FD8CD868	Beep
00000015	FD89E730	00000000	00000000	00000000	FD89C968	NwlnkIpx
00000001	FD892170	00000000	00000000	00000000	FD8980E8	Netbios
00000000	FD89D730	00000000	00000000	00000000	FD897D68	Ip
00000001	FD8CBB70	00000000	00000000	FD8DAA08	FD8CAF88	KeyboardClass0
00000001	FD8C9F30	00000000	00000000	00000000	FD8C60F0	Video0
00000001	FD8C9C90	00000000	00000000	00000000	FD8C50F8	Video1
00000001	FD8CC530	00000000	00000000	FD8DAC08	FD8CBF88	PointerClass0
00000001	FD8DB550	FD8D3030	00000000	00000000	FD8D3FC8	RawTape
00000007	FD89D730	FD897CB0	00000000	00000000	FD897C48	Tcp
00000001	FD88A990	00000000	00000000	00000000	FD88A8A8	ParallelPort0
00000003	FD8B3730	00000000	00000000	00000000	FD8A40E8	NE20001

This example uses the DEVICE command with the BEEP device object's name.

DEVICE beep

```
RefCnt   DrvObj   NextDev  AttDev   CurIrp   DevExten Name
00000000 FD8CD910 00000000 00000000 00000000 FD8CD868 Beep
Timer*               : 00000000
Flags                : 00000044 DO_BUFFERED_IO | DO_DEVICE_HAS_NAME
Characteristics      : 00000000
Vpb*                 : 00000000
Device Type          : 1        FILE_DEVICE_BEEP
StackSize            : 1
&Queue               : FD8CD7E4
AlignmentRequirement: 00000000 FILE_BYTE_ALIGNMENT
&DeviceQueue         : FD8CD810
&Dpc                 : FD8CD824
ActiveThreadCount    : 00000000
SecurityDescriptor* : E10E2528
&DeviceLock          : FD8CD84C
SectorSize           : 0000
Spare1               : 0000
DeviceObjectExtn*    : FD8CD8B8
Reserved*            : 00000000
```

DEX

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Display or assign a Data window expression.

Syntax

DEX [*data-window-number* [*expression*]]

data-window-number	Number from 0 to 3 indicating which Data window to use. This number displays on the righthand side of the line above the Data window.

Use

The DEX command assigns a data expression to any of the four SoftICE Data windows. Every time SoftICE pops up, the expressions are re-evaluated and the memory at that location displays in the appropriate Data window. This is useful for displaying changing memory locations where there is always a pointer to the memory in either a register or a variable. The data displays in the current format of the Data window: either byte, word, dword, short real, long real, or 10-byte real. This command is the same as entering the command D expression every time SoftICE pops up.

If you type DEX without parameters, it displays all the expressions currently assigned to the Data windows.

To unassign an expression from a Data window, type DEX followed by the data-window-number, then press Enter.

To cycle through the four Data windows, use the DATA command. Refer to *DATA* on page 63.

Example

Every time SoftICE pops up, Data window 0 contains the contents of the stack.

DEX 0 ss:esp

Every time SoftICE pops up, Data window 1 contains the contents of the memory pointed at by the public variable PointerVariable.

DEX 1 @pointervariable

See Also

DATA

DIAL

Windows 95, Windows 98, Windows NT *Customization*

Redirect console to modem.

Syntax

DIAL [**on** [***com-port***] [***baud-rate***] [**i=*init-string***] [**p=*number***] | **off**]

com-port	If no com-port is specified it uses COM1.
baud-rate	Baud-rate to use for modem communications. The default is 38400. The rates are 1200, 2400, 4800, 9600, 19200, 23040, 28800, 38400, 57000, 115000.
i=init-string	Optional modem initialization string.
p=number	Telephone number.

Use

The DIAL command initiates a call to a remote machine via a modem. The remote machine must be running SERIAL.EXE and be waiting for a call. Once a connection is established, SoftICE input is received from the remote machine and SoftICE output is sent to the remote machine. No input is accepted from the local machine except for the pop-up hot key sequence.

You can specify the modem initialization string and phone number within the SoftICE configuration settings, so that the strings they specify become the defaults for the i and p command-line parameters. Refer to Chapter 8, "Customizing SoftICE" in the *Using SoftICE* manual.

On the remote machine, only the com-port, baud-rate, and init parameters should be specified to SERIAL.EXE.

Example

The following is an example of the DIAL command:

```
DIAL on 2 19200 i=atx0 p=9,555-5555,,,1000
```

The command tells SoftICE to first initialize the modem on com-port 2 with the string, "atx0," and then to make a call through the modem to the telephone number 9-555-5555 extension 1000. Commas can be used in the phone number, just as with traditional modem software, to insert delays into the dialing sequence.

The following example shows the syntax expected by SERIAL.EXE when running it on a remote machine so that it answers a DIAL command from the local machine:

SERIAL on [*com-port*] [*baud-rate*] i"*init-string*"

The following SERIAL.EXE command-line uses a modem initialization string of "atx0" to answer a call (at 19200 bps) through a modem on the remote machine's COM1 serial port. The command line is entered on the remote machine.

```
SERIAL on 1 19200 i"atx0"
```

When the remote debugging session is complete, enter the DIAL OFF command from the remote machine to terminate the debugging session and hang up the modem.

The following are examples of the Dial initialization and Phone number strings in the Remote Debugging SoftICE configuration settings:

```
Dial initialization string:  atx0
Telephone number string:  9,555-5555,,,1000
```

With the Dial initialization string in place, SoftICE always initializes the modem specified in DIAL commands with "ATX0", unless the DIAL command explicitly specifies an initialization string.

With the Phone initialization string in place, SoftICE always dials the specified number when executing DIAL commands, unless the DIAL command explicitly specifies a phone number.

See Also

ANSWER, SERIAL, and Chapter 7, "Debugging Remotely," in the *Using SoftICE* manual.

DRIVER

Windows 98, Windows NT *System Information*

Display information on Windows NT drivers.

Syntax

DRIVER [***driver-name*** | ***pdriver-object***]

Use

The DRIVER command displays information on Windows NT drivers. If the DRIVER command is entered without parameters, summary information is shown for all drivers found in the \Driver directory. However, if a specific driver is indicated, either by its object directory name (driver-name), or by its object address (pdriver-object), more detailed information is displayed.

If a directory is not specified with the driver-name, the DRIVER command attempts to locate the named driver in the \Driver object directory. To display information about a driver that is not located in the \Driver directory, you must specify the complete object path name of the driver.

When displaying detailed information about a specified driver, the DRIVER command displays the fields of the DRIVER_OBJECT data structure as defined in NTDDK.H.

Output

The following fields are shown as summary information:

Start	Base address of the driver.
Size	Driver's image size.
DrvSect	Pointer to driver module structure.
Count	Number of times the registered reinitialization routine has been invoked for the driver.
DrvInit	Address of the driver's DriverEntry routine.
DrvStaIo	Address of the driver's StartIo routine.
DrvUnld	Address of the driver's Unload routine.
Name	Name of the driver.

The following is shown when detailed information is printed:

DeviceObject	Pointer to the first device object on the driver's linked list of device objects that it owns.
Flags	Field is a bit-mask of driver flag. The only flag currently documented is DRVO_UNLOAD_INVOKED.

FastIoDispatch	Pointer to the driver's fast I/O dispatch data structure, if it has one. File System Drivers typically have a fast I/O routines defined for them. Information on the structure can be found in NTDDK.H.
Handler Addresses	Upon initialization, driver's can register handlers that are called when the driver receives specific IRP request types. Each handler address is listed along with the IRP major function it processes for the driver.

Example

The following example shows the output of the DRIVER command with no parameters. This results in SoftICE printing summary information on all the drivers in the \Driver object directory.

DRIVER

Start	Size	DrvSect	Count	DrvInit	DrvStaIo	DrvUnld	Name
FB030000	00000E20	FD8CDA88	00000000	FB0302EE	FB0305E8	FB0306E2	Beep
FB130000	0000D3A0	FD89E8C8	00000000	FB13B7BF	00000000	FB136789	NwlnkIpx
FB050000	00002320	FD8CD1A8	00000000	FB050AF2	FB0508BE	00000000	Mouclass
FB060000	00002320	FD8CBC48	00000000	FB060AF2	FB0608C0	00000000	Kbdclass
FB070000	00003860	FD8CAE48	00000000	FB070B0C	00000000	00000000	VgaSave

The following is an example of the DRIVER command with the BEEP.SYS driver object's name as a parameter. From the listing it can be seen that the driver's first device object is at FD8CD7B0h, and that it has 4 IRP handler routines registered.

DRIVER beep

```
Start    Size     DrvSect  Count    DrvInit  DrvStaIo DrvUnld  Name
FB030000 00000E20 FD8CDA88 00000000 FB0302EE FB0305E8 FB0306E2 Beep
DeviceObject*    : FD8CD7B0
Flags            : 00000000
HardwareDatabase : \REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM
FastIoDispatch*  : 00000000
IRP_MJ_CREATE                        at 8:FB03053C
IRP_MJ_CLOSE                         at 8:FB03058A
IRP_MJ_DEVICE_CONTROL                at 8:FB0304C6
IRP_MJ_CLEANUP                       at 8:FB030416
```

E

Windows 3.1, Windows 95, Windows 98, Windows NT *Display/Change Memory*

Edit memory.

Syntax

`E[`*`size`*`] [`*`address`*` [`*`data-list`*`]]`

size

Value	Description
B	Byte
W	Word
D	Double Word
S	Short Real
L	Long Real
T	10-Byte Real

data-list List of data objects of the specified size (bytes, words, double words, short reals, long reals, or 10-byte reals) or quoted strings separated by commas or spaces. The quoted string can be enclosed with single quotes or double quotes.

Use

If you do not specify data-list, the cursor moves into the Data window where you can edit the memory in place. If you specify a data-list, the memory is immediately changed to its new values.

If the Data window is not currently visible, it is automatically made visible. Both ASCII and hexadecimal edit modes are supported. To toggle between the ASCII and hexadecimal display areas, press the Tab key.

If you do not specify a size, the last size used is assumed.

Enter valid floating point numbers in the following format:

`[`*`leading sign`*`]` *`decimal-digits`* `.` *`decimal-digits`* `E` *`sign exponent`*

Example: A valid floating point number is -1.123456 E-19

Example

The following command moves the cursor into the Data window for editing. The starting address in the Data window is at DS:1000h, and the data displays in hexadecimal byte format as well as in ASCII. The initial edit mode is hexadecimal.

```
EB ds:1000
```

The next command moves the null terminated ASCII string 'Test String' into memory at location DS:1000h.

```
EB ds:1000 'Test String',0
```

This command moves the short real number 3.1415 into the memory location DS:1000h.

```
ES ds:1000 3.1415
```

EC

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

F6

Enter or exit the Code window.

Syntax

EC

Use

The EC command toggles the cursor between the Code window and the Command window:

- If the cursor is in the Command window, it moves to the Code window.
- If the cursor is in the Code window, it moves to the Command window.
- If the Code window is not visible when the command is entered, it is made visible.

When the cursor is in the Code window, several options become available that make debugging much easier. These options are as follows:

- Set point-and-shoot breakpoints
  Set these with the BPX command. If you do not specify parameters with the BPX command (default key F9), an execution breakpoint is set at the location of the cursor position in the Code window.
- Go to cursor line
  Set a temporary breakpoint at the cursor line and begin executing with the HERE command (default key F7).
- Scroll the Code window
  The scrolling keys (UpArrow, DownArrow, PageUp and PageDn) are redefined while the cursor is in the Code window:
  - ◊ UpArrow: Scroll Code window up one line.
  - ◊ DownArrow: Scroll Code window down one line.
  - ◊ PageUp: Scroll Code window up one window.
  - ◊ PageDn: Scroll Code window down one window.

Source Mode Only

Scroll the Code window from the Command window using the CTRL key with one of the previously mentioned cursor keys. The following keys also have special meaning:

- CTRL-Home: Moves to line 1 of current source file.
- CTRL-End: Moves to the last line of the current source file.

Note: The previous keys only work for source display, not for disassembled instructions.

- CTRL-RightArrow: Horizontal scroll of source code right.
- CTRL-LeftArrow: Horizontal scroll of source code left.

EXIT

Windows 3.1 *Flow Control*

Force an exit of the current DOS or Windows program.

Syntax

EXIT

Use

The EXIT command attempts to abort the current DOS or Windows program by forcing a DOS exit function (INT 21h, function 4Ch). This command only works if DOS is in a state where it is able to accept the exit function call. If this call is made from certain interrupt routines, or other times when DOS is not ready, the system may behave unpredictably. Only use this call when SoftICE pops up in VM mode or 16- or 32-bit protected mode running at ring 3. In 32-bit, ring 0 protected mode code, an error displays.

Caution

Use the EXIT command with care. Because SoftICE can be popped up at any time, a situation can occur where DOS is not in a state to accept an exit function call. Also, the EXIT command does not have any program-specific resetting.

Example: The EXIT command does not reset the video mode or interrupt vectors. For Windows programs, the EXIT command does not free resources.

If running under WIN32s, the EXIT command sometimes causes WIN32s to pop up with an unhandled exception occurred dialog box. Press OK to terminate the application.

For Windows 95 and Windows NT

EXIT is no longer supported.

Example

Causes the current DOS or Windows program to exit.

EXIT

EXP

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

Display export symbols from DLLs.

Syntax

EXP [[**module!**][***partial-name***]] | [**!**]

module!	Display exports from the specified module only.
partial-name	Export symbol or the first few characters of the name of an export symbol name. The ? character can be used as a wildcard character in place of any character in the export name.
!	Display list of modules for which SoftICE has exports loaded.

Use

Use the EXP command to show exports from Windows DLLs, Windows NT drivers, and 16-bit drivers (.DRV extension) for which SoftICE has exports loaded. To tell SoftICE which DLLs and drivers to load, set the SoftICE initialization settings for Exports in Symbol Loader.

The module and name parameters can be used to selectively display exports only from the specified module, and/or exports that match the characters and wildcards in the name parameter. When exports are displayed, the module name is printed first on a line by itself, and the export names are printed below it, along with their addresses.

Note: Since DLLs and drivers run in protected mode, the addresses are protected mode addresses.

This command is valid for both 16-bit and 32-bit DLLs with 16-bit exports being listed first.

For Windows 3.1

SoftICE automatically loads exports for KERNEL, USER, and GDI.

For Windows 95

SoftICE automatically loads exports for KERNEL, USER, and GDI. The SoftICE Loader can dynamically load 32-bit exported symbols.

For Windows NT

SoftICE automatically loads exports for KERNEL32, USER32, and GDI32. The SoftICE loader can dynamically load 32-bit exported symbols.

Example

The following example of the EXP command being used to display all exports that begin with the string DELETE: The output shows that KERNEL.DLL has 3 exports matching the string: DELETEATOM, DELETEFILE, and DELETEPATHNAME. These routines are located at 127:E3, 11F:7D4 and 127:345A, respectively. Following the exports from KERNEL are the exports from USER and GDI, and following these begin the 32-bit exports.

:EXP delete

```
KERNEL
   0127:00E3 DELETEATOM           011F:07D4 DELETEFILE
   0127:345A DELETEPATHNAME

USER
   176F:0C88 DELETEMENU

GDI
   0527:0000 DELETEMETAFILE       04B7:211C DELETESPOOLPAGE
   047F:55FD DELETEDC             054F:0192 DELETEPQ
   047F:564B DELETEOBJECT         04B7:226E DELETEJOB
   0587:A22E DELETEENHMETAFILE

KERNEL32
0137:BFF97E9B DeleteAtom          0137:BFF88636 DeleteCriticalSection
0137:BFF9DC5A DeleteFileA         0137:BFFA4C49 DeleteFileW

USER32
0137:BFF62228 DeleteMenu
GDI32
0137:BFF3248F DeleteColorSpace    0137:BFF32497 DeleteDC
0137:BFF3248B DeleteEnhMetaFile   0137:BFF31111 DeleteMetaFile
0137:BFF3249F DeleteObject
```

In the following example, the ! character is used to narrow EXP's output to only those modules which are listed to the left of the !. In the case where no DLL or driver is specified before the !, SoftICE simply dumps the names of all the modules for which it has exports loaded.

:EXP !

```
KERNEL
USER
GDI
KERNEL32
USER32
GDI32
```

The next example is of the EXP command being used to list all exports within USER32.DLL that start with "IS." The ! character is used here to differentiate the module name from the name qualifier.

```
:EXP user32!is

USER32
0137:BFF64290 IsCharAlphaA
0137:BFF64256 IsCharAlphaNumericA
0137:BFF61014 IsCharAlphaNumericW
0137:BFF61014 IsCharAlphaW
0137:BFF641E8 IsCharLowerA
0137:BFF61014 IsCharLowerW
0137:BFF64222 IsCharUpperA
0137:BFF61014 IsCharUpperW
0137:BFF61F6A IsChild
0137:BFF6480F IsClipboardFormatAvailable
0137:BFF64D7C IsDialogMessage
0137:BFF64D7C IsDialogMessageA
0137:BFF6101D IsDialogMessageW
0137:BFF618A4 IsDlgButtonChecked
0137:BFF62F12 IsHungThread
0137:BFF64697 IsIconic
0137:BFF623A5 IsMenu
0137:BFF649B9 IsRectEmpty
0137:BFF644BF IsWindow
0137:BFF646E1 IsWindowEnabled
0137:BFF638C4 IsWindowUnicode
0137:BFF64706 IsWindowVisible
0137:BFF646BC IsZoomed
```

See Also

SYMBOL, TABLE

F

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Fill memory with data.

Syntax

F *address* l *length data-list*

length	Length in bytes.
data-list	List of bytes or quoted strings separated by commas or spaces. A quoted string can be enclosed with single quotes or double quotes.

Use

Memory is filled with the series of bytes or characters specified in the data-list. Memory is filled starting at the specified address and continues for the specified length. If the data-list length is less than the specified length, the data-list is repeated as many times as necessary.

Example

Fills memory starting at location DS:8000h for a length of 100h bytes with the string 'Test'. The string 'Test' is repeated until the fill length is exhausted.

```
F ds:8000 l 100 'test'
```

FAULTS

Windows 3.1, Windows 95, Windows 98, Windows NT *Mode Control*

Turn fault trapping on or off.

Syntax

FAULTS [on | off]

Use

Use the FAULTS command to turn SoftICE processor fault trapping on or off.

Example

Turns off fault trapping in SoftICE.

```
FAULTS off
```

See Also

SET

FIBER

Windows NT *System Information*

Dump a fiber data structure.

Syntax

FIBER [*address*]

Use

Use the FIBER command to dump a fiber data structure returned by CreateFiber(). If you do not specify an address, FIBER dumps the fiber data associated with the current thread. SoftICE provides a stack trace after the dump.

Example

The following example dumps the fiber data associated with the current thread:

:FIBER

```
Fiber state for the current thread:
  User data:004565D0  SEH Ptr:01C2FFA8
  Stack top:01C30000   Stack bottom:01C2F000  Stack limit:01B30000
  EBX=00000001  ESI=005862B8  EDI=004565D0  EBP=01C2FF88  ESP=01C2FC4C
  EIP=63011BAF a.k.a. WININET!.text+00010BAF

=> at 001B:00579720
```

FILE

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

Change or display the current source file.

Syntax

FILE [[*****]***file-name***]

Use

The FILE command is often useful when setting a breakpoint on a line that has no associated symbol. Use FILE to bring the desired file into the Code window, use the SS command to locate the specific line, move the cursor to the specific line, then enter BPX or press F9 to set the breakpoint.

- If you specify file-name, that file becomes the current file and the start of the file displays in the Code window.
- If you do not specify file-name, the name of the current source file, if any, displays.
- If you specify the * (asterisk), all files in the current symbol table display.

Only source files that are loaded into memory with Symbol Loader or are pre-loaded at initialization are available with the FILE command.

For Windows 95 and Windows NT

Specifying the FILE file-name command switches address contexts within SoftICE, if the current symbol table has an associated address context.

Example

If main.c is loaded with the SoftICE Loader, this command displays it in the Code window starting with line 1.

```
FILE main.c
```

FKEY

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Show and edit the function key assignments.

Syntax

FKEY [***function-key string***]

function-key

Key	Description
F1 - **F12**	Unshifted function key
SF1 - **SF12**	Shifted function key
CF1 - **CF12**	Control key plus function key
AF1 - **AF12**	Alternate key plus function key

string Consists of any valid SoftICE commands and the special characters caret (^) and semicolon (;). Place a caret (^) at the beginning of a command to make the command invisible. Place a semicolon (;) in the string in place of Enter.

Use

Use the FKEY command to assign a string of one or more commands to a function-key. If you do not specify parameters, the current function-key assignments display.

Hint: You can also edit function key assignments by modifying the SoftICE initialization settings for Keyboard Mappings in Symbol Loader. Refer to the *Using SoftICE* manual for more information about customizing SoftICE.

To unassign a specified function-key, use the FKEY command with the parameters function_key_name followed by null_string.

Use carriage return symbols in a function-key assignment string to assign a function-key a series of commands. A carriage return is represented by a semi-colon (;).

If you put a caret "^" or press Shift-6 in front of a command name, the subsequent command becomes invisible. The command functions as normal, but all information that normally displays in the Command window (excluding error messages) is suppressed. The invisible mode is useful when a command changes information in a window (Code, Register, or Data), but you do not want to clutter the Command window.

SoftICE implements the function-keys by inserting the entire string into its keyboard buffer. The function-keys can therefore be used anyplace where a valid command can be typed. If you want a function key assignment to be in effect every time you use SoftICE, pre-initialize the keyboard mappings within your SOFTICE configuration settings. Refer to Chapter 8, "Customizing SoftICE" in the *Using SoftICE* guide.

Example

This example assigns the toggle Register window command to the F2 function-key. The caret "^" makes the function invisible, and the semicolon ";" ends the function with a carriage return. After you enter this command, press the F2 key to toggle the Register window on or off.

FKEY f2 ^wr;

The next example shows that multiple commands can be assigned to a single function and that partial commands can be assigned for the user to complete. After you enter this command, pressing the Ctrl F1 key sequence causes the program to execute until location CS:8028F000h is reached, displays the stack contents, and starts the U command for the user to complete.

FKEY cf1 g cs:8028f000;d ss:esp;u cs:eip+

After you enter this example, pressing the F1 key makes the Data window three lines long and dumps data starting at 100h in the segment currently displayed in the Data window.

FKEY f1 wd 3;d 100;

The following example toggles the Register window, and creates a Locals window of length 8 and a Code window of length 10.

FKEY f1 wr;wl 8;wc 10;

FOBJ

Windows NT *System Information*

Display information about a file object.

Syntax

FOBJ [***fobj-address***]

fobj-address Address of the start of the file object structure to be displayed.

Use

The FOBJ command displays the contents of kernel file objects. The command checks for the validity of the specified file object by insuring that the device object referenced by it is a legitimate device object.

The fields shown by SoftICE are not documented in their entirety here, as adequate information about them can be found in NTDDK.H in the Windows NT DDK. A few fields deserve special mention, however, because device driver writers find them particularly useful:

DeviceObject This field is a pointer to the device object associated with the file object.

Vpb This is a pointer to the volume parameter block associated with the file object (if any).

FSContext1 and FSContext2 These are file system driver (FSD) private fields that can serve as keys to aid the driver in determining what internal FSD data is associated with the object.

Other fields of interest, whose purpose should be fairly obvious, include the access protection booleans, the Flags, the FileName and the CurrentByteOffset.

Example

The following example shows the FOBJ command's output:

:FOBJ fd877230

```
DeviceObject *    : FD881570
Vpb *             : 00000000
FsContext *       : FD877188
FsContext2 *      : FD877C48
SecObjPointer *   : FD8771B4
PrivateCacheMap * : 00000001
FinalStatus       : 00000000
RelatedFileObj *  : 00000000
LockOperation     : False
DeletePending     : False
ReadAccess        : True
```

```
WriteAccess       : True
DeleteAccess      : False
SharedRead        : True
SharedWrite       : True
SharedDelete      : False
Flags             : 00040002 FO_SYNCHRONOUS_IO | FO_HANDLE_CREATED
FileName          : \G:\SS\data\status.dat
CurrentByteOffset : 00
Waiters           : 00000000
Busy              : 00000000
LastLock*         : 00000000
&Lock             : FD877294
&Event            : FD8772A4
ComplContext*     : 00000000
```

FLASH

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Restore the Windows screen during P and T commands.

Syntax

FLASH [on | off]

Use

Use the FLASH command to specify whether the Windows screen restores during any T (trace) and P (step over) commands. If you specify that the Windows screen is to be restored, it is restored for the brief time period that the P or T command is executing. This feature is needed to debug sections of code that access video memory directly.

If the routine being called writes to the Windows screen and if the P command executes across a call, the screen restores. When debugging protected mode applications such as VxDs or Windows applications with FLASH off, this is generally not the case. SoftICE restores the screen only if the display driver is called before the call is completed.

If you do not specify a parameter, the current state of FLASH displays.

The default is FLASH OFF.

Example

This command turns on FLASH mode. The Windows screen restores during any subsequent P or T commands.

```
FLASH on
```

See Also

SET

FORMAT

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Shift-F3

Change the format of the Data window.

Syntax

`FORMAT`

Use

Use the FORMAT command to change the display format in the currently displayed Data window. Change the formats in the order byte, word, dword, short real, long real, 10-byte real, and then starting back at byte. This command is most useful when assigned to a function key. The default function key assignment is Shift-F3. The Shift-F3 is also supported when editing in the Data window.

Example

Changes the Data window to the next data format.

`FORMAT`

G

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

Go to an address.

Syntax

G [*=start-address*] [*break-address*]

=start-address	Any expression that resolves to a valid address is acceptable.
break-address	Any expression that resolves to a valid address is acceptable.

Use

The G command exits from SoftICE. If you specify break-address, a single one-time execution breakpoint is set on that address. In addition, all sticky breakpoints are armed.

Execution begins at the current CS:EIP unless you supply the start-address parameter. If you supply the start-address parameter, execution begins at start-address. Execution continues until the break-address is encountered, the SoftICE pop-up key sequence is used, or a sticky breakpoint is triggered. When SoftICE pops up, for any reason, the one-time execution breakpoint is cleared.

The break-address must be the first byte of an instruction opcode.

The G command without parameters behaves the same as the X command.

If the Register window is visible when SoftICE pops up, all registers that have been altered since the G command was issued are displayed with the bold video attribute.

For Windows 3.1

The non-sticky execution breakpoint uses an INT 3 style breakpoint.

For Windows 95 and Windows NT

The non-sticky execution breakpoint uses debug registers unless none are available. If none are available, it uses INT 3.

Example

This command sets a one-time breakpoint at address CS:80123456h.

```
G 80123456
```

GDT

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the Global Descriptor Table.

Syntax

GDT [***selector***]

selector	Starting GDT selector to display

Use

The GDT command displays the contents of the Global Descriptor Table. If you specify an optional selector, only information on that selector is listed. If the specified selector is an LDT selector (bit 2 is a 1), SoftICE automatically displays information from the LDT, rather than the GDT.

Output

The base linear address and limit of the GDT is shown at the top of the GDT command's output. Each subsequent line of the output contains the following information:

selector value Lower two bits of this value reflects the descriptor privilege level.

selector type One of the following:

Type	Description
Code16	16-bit code selector
Data16	16-bit data selector
Code32	32-bit code selector
Data32	32-bit data selector
LDT	Local Descriptor Table selector
TSS32	32-bit Task State Segment selector
TSS16	16-bit Task State Segment selector
CallG32	32-bit Call Gate selector
CallG16	16-bit Call Gate selector
TaskG32	32-bit Task Gate selector
TaskG16	16-bit Task Gate selector
TrapG32	32-bit Trap Gate selector

Type	Description
TrapG16	16-bit Trap Gate selector
IntG32	32-bit Interrupt Gate selector
IntG16	16-bit Interrupt Gate selector
Reserved	Reserved selector

selector base Linear base address of the selector.

selector limit Size of selector's segment.

selector DPL Selector's descriptor privilege level (DPL), which is either 0, 1, 2 or 3.

present bit P or NP, indicating whether the selector is present or not present.

segment attributes One of the following:

Value	Description
RW	Data selector is readable and writable.
RO	Data selector is read only.
RE	Code selector is readable and executable.
EO	Code selector is execute only.
B	TSS's busy bit is set.
ED	Expand down data selector.

Example

The following command shows abbreviated output from the GDT command.

:GDT

```
Sel.  Type      Base      Limit     DPL  Attributes
GDTbase=C1398000  Limit=0FFF
0008  Code16    00017370  0000FFFF  0    P   RE
0010  Data16    00017370  0000FFFF  0    P   RW
0018  TSS32     C000AEBC  00002069  0    P   B
0020  Data16    C1398000  00000FFF  0    P   RW
0028  Code32    00000000  FFFFFFFF  0    P   RE
0030  Data32    00000000  FFFFFFFF  0    P   RW
003B  Code16    C33E9800  000007FF  3    P   RE
0043  Data16    00000400  000002FF  3    P   RW
0048  Code16    00013B10  0000FFFF  0    P   RE
0050  Data16    00013B10  0000FFFF  0    P   RW
0058  Reserved  00000000  0000FFFF  0    NP
0060  Reserved  00000000  0000FFFF  0    NP
0068  TSS32     C0015DE8  00000068  0    P
```

GENINT

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

Force an interrupt to occur.

Syntax

GENINT [nmi | int1 | int3 | *interrupt-number*]

interrupt-number	For Windows 3.1 and Windows 95: Valid interrupt number between 0 and 5Fh. For Windows NT: Valid interrupt number between 0 and FFh.

Use

The GENINT command forces an interrupt to occur. Use this function to hand off control to another debugger you are using with SoftICE. Also use it to test interrupt routines.

The GENINT command simulates the processing sequence of a hardware interrupt or an INT instruction. It vectors control through the current IDT entry for the specified interrupt number.

Warning: Ensure that there is a valid interrupt handler before using this command. SoftICE does not know if there is a handler installed. Your machine will most likely crash if there is not one.

GENINT cannot be used to simulate a processor fault that pushes an exception code. For example, GENINT cannot simulate a general protection fault.

Example

The following command forces a non-maskable interrupt. It gives control back to CodeView for DOS, if you use SoftICE as an assistant to CodeView for DOS.

```
GENINT nmi
```

If using CodeView for Windows, use the command:

```
GENINT 0
```

For other debuggers, experiment with interrupt-numbers 0, 1, 2 and 3.

When the command I3HERE==ON, and you are using a level -3 debugger, such as BoundsChecker, SoftICE traps on any INT 3 breakpoints installed by the level-3 debugger. When this happens, set I3HERE==OFF, and use the GENINT command to reactivate the breakpoint. This returns control to the level -3 debugger, and SoftICE does not trap subsequent INT 3s.

```
I3HERE off
GENINT 3
```

H

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

F1

Display help information.

Syntax

For Windows 3.1

H [***command*** **|** ***expression***]

For Windows 95 and Windows NT

H [***command***]

Use

For Windows 3.1

Under Windows 3.1, the parameter you supply determines whether help is displayed or an expression is evaluated. If you specify a command, help displays detailed information about the command, including the command syntax and an example. If you specify an expression, the expression is evaluated, and the result is displayed in hexadecimal, decimal, signed decimal (only if < 0), and ASCII.

For Windows 95 and Windows NT

Under Windows 95 and Windows NT, the H command displays help on SoftICE commands. (Refer to *?* on page 3 for information about evaluating expressions under Windows 95 and Windows NT.) To display general help on all the SoftICE commands, enter the H command with no parameters. To see detailed information about a specific command, use the H command followed by the name of the command on which you want help. Help displays a description of the command, the command syntax, and an example.

Example

The following example displays information about the ALTKEY command:

```
:H altkey
```

```
Set key sequence to invoke window
ALTKEY [ALT letter | CTRL letter]
ex: ALTKEY ALT D
```

See Also

?

HBOOT

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

Do a hard system boot (total reset).

Syntax

HBOOT

Use

The HBOOT command resets the computer system. SoftICE is not retained in the reset process. HBOOT is sufficient unless an adapter card requires a power-on reset. In those rare cases, the machine power must be recycled.

HBOOT performs the same level of system reset as pressing Ctrl-Alt-Delete when not in SoftICE.

Example

To make the system reboot, use this command:

HBOOT

HEAP

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the Windows global heap.

Syntax

HEAP -L [free | *module-name* | *selector*]

-L	Display only global heap entries that contain a local heap.
module-name	Name of the module.
selector	LDT selector.

Use

For Windows 95

For 16-bit modules, the HEAP command works the same as it does under Windows 3.1.

For Windows NT

For 16-bit modules, the HEAP command works the same as it does under Windows 3.1, but is process-specific. You must be in a NTVDM process that contains a WOW (Windows on Windows) box.

For Windows 95, refer to HEAP32 on page 97.

For Windows NT, refer to HEAP32 on page 100.

For Windows 3.1

The HEAP command displays the Windows global heap in the Command window.

- If you do not specify parameters, the entire global heap displays.
- If you specify FREE, only heap entries marked as FREE display.
- If you specify the module name, only heap entries belonging to the module display.
- If you specify an LDT selector, only a single heap entry corresponding to the selector displays.

At the end of the listing, the total amount of memory used by the heap entries that displayed is shown. If the current CS:EIP belongs to one of the heap entries, that entry displays with the bold video attribute.

If there is no current LDT, the HEAP command is unable to display heap information.

Output

For each heap entry the following information displays:

selector or handle — In Windows 3.1, this is almost the same thing. Heap selectors all have a dpl of 3 while the corresponding handle is the same selector with a dpl of 2. For example, if the handle was 106h the selector would be 107h. Use either of these in an expression.

address — 32-bit flat virtual address.

size — Size of the heap entry in bytes.

module name — Module name of the owner of the heap entry.

type — Type of entry. One of the following:

Type	Description
Code	Non-discardable code segment
Code D	Discardable code segment
Data	Data segment
ModuleDB	Module data base segment
TaskDB	Task data base segment
BurgerM	Burger Master (The heap itself)
Alloc	Allocated memory
Resource	Windows Resource

Additional Type Information

If the heap entry is a code or a data segment, the segment number from the .EXE file displays. If the heap entry is a resource, one of the following resource types may display:

`UserDef`	`Icon`	`String`	`Accel`	`IconGrp`
`Cursor`	`Menu`	`FontGrp`	`ErrTable`	`NameTabl`
`Bitmap`	`Dialog`	`Font`	`CursGrp`	

Example

To display all heap entries belonging to the KERNEL module, use the following command:

HEAP kernel

Han/Sel	Address	Length	Owner	Type	Seg/Rsr
00F5	000311C0	000004C0	KERNEL	ModuleDB	
00FD	00031680	00007600	KERNEL	Code	01
0575	00054220	00003640	KERNEL	Alloc	
0106	00083E40	00002660	KERNEL	Code D	02
010E	805089A0	00001300	KERNEL	Code D	03
0096	80520440	00000C20	KERNEL	Alloc	

Total Memory:62K

See Also

For Windows 95, refer to *HEAP32* on page 97.
For Windows NT, refer to *HEAP32* on page 100.

HEAP32

Windows 95, Windows 98 *System Information*

Display the Windows global heap.

Syntax

HEAP32 [***hheap32*** | ***task-name***]]

hheap32	Heap handle returned from HeapCreate.
task-name	Name of any 32-bit task.

Use

For Windows 95

The HEAP32 command displays heaps for a process.

Note: For 16-bit modules, use the *HEAP32* on page 100.

The HEAP32 command displays the following:

- KERNEL32 default system heap.
- Private heaps of processes created through the HeapCreate( ) function.
- Two Ring-0 heaps created by VMM. The first one displayed is the pagelocked heap, and the second is the pagetable heap.
- One Ring-0 heap for every existing virtual machine.

For Windows 3.1, Windows 95, and Windows NT, refer to HEAP on page 94.

For Windows NT, refer to HEAP32 on page 100.

If you provide a process name, SoftICE displays the entire default process heap for that process, and the address context automatically changes to that of the process. To view a nondefault heap for a process, specify the heap base address instead of the process name.

The debug versions of Windows 95 provide extra debugging information for each heap element within a heap. To see this information, you must be running the appropriate debug version, as follows:

- For KERNEL32 Ring-3 heaps, have the SDK debug version installed.
- For VMM Ring-0 heaps, have the DDK debug version of VMM installed.

Output

For each heap entry, the following information displays:

HeapBase	Address where the heap begins.
MaxSize	Current maximum size the heap can grow without creating a new segment.
Committed	Number of kilobytes of committed memory that are currently present in physical memory.

Segments — Number of segments in the heap. Each time the heap grows past the current maximum size, a new heap segment is created.

Type

Heap Type	Description
Private	Ring 3 heap created by an application process.
System	Ring 3 default heap for KERNEL32.
Ring0	Ring 0 heap created by VMM.
VM##	Heap created by VMM for a specific Virtual Machine to hold data structures specific to that VM.

Owner — Name of the process that owns the heap.

When displaying an individual 32-bit heap, the following information displays:

Heap Type	Description
Address	Address of the heap element
Size	Size in bytes of the heap element
Free	If the heap element is a free block, the word FREE appears; otherwise, the field is blank.

With the appropriate debug versions of the SDK and DDK, the following extra information appears for each heap element:

Heap Element	Description
EIP	EIP address of the code that allocated the heap element.
TID	VMM thread-id of the allocating thread
Owner	Nearest symbol to the EIP address

Example

To display all 32-bit heaps, use the command:

HEAP32

HeapBase	Max Size	Commit-ted	Seg-ments	Type	Owner
00EA0000	1024K	8K	1	Private	Mapisp32
00DA0000	1024K	8K	1	Private	Mapisp32
00CA0000	1024K	8K	1	Private	Mapisp32
00960000	1024K	8K	1	Private	Mapisp32
00860000	1024K	8K	1	Private	Mapisp32

To display all heap entries for Exchng32, use the command:

HEAP32 exchng32

```
Heap: 00400000    Max Size: 1028K   Committed: 12K    Segments: 1
Address           Size
00400078          000004E4
00400560          00000098
004005FC          00000054
00400654          000000A4
004006FC          00000010
00400710          00000014          Free
```

See Also

For Windows 3.1, Windows 95, and Windows NT, refer to *HEAP* on page 94.
For Windows NT, refer to *HEAP32* on page 100.

HEAP32

Windows NT *System Information*

Display the Windows heap.

Syntax

HEAP32 [[**-w -x -s -v -b -trace**] [***heap*** | ***heap-entry*** | ***process-type***]]

-w	Walk the heap, showing information about each heap entry.
-x	Show an extended summary of a 32-bit heap.
-s	Provide a segment summary for a heap.
-v	Validate a heap or heap-entry.
-b	Show base address and sizes of heap entry headers.
-trace	Display a heap trace buffer.
heap	32-bit heap handle.
heap-entry	Heap allocated block returned by HeapAlloc or HeapRealloc.
process-type	Process name, process-id, or process handle (KPEB).

Use

All HEAP32 options and parameters are optional. If you do not specify options or parameters, a basic heap summary displays for every heap in every process. If a parameter is specified without options, a summary will be performed for the heap-entry, heap, or in the case of a process-type, a summary for each heap within the process.

Note: All 16-bit HEAP functionality still works. Refer to *HEAP* on page 94 for Windows 3.1. This information only applies to HEAP32.

For Windows 3.1, Windows 95, and Windows NT, refer to HEAP on page 94.

For Windows 95, refer to HEAP32 on page 97.

The -Walk option walks a heap, showing the state of each heap-entry on a heap. The Walk option is the default option if you specify a heap handle without other options.

The -eXtended option displays a detailed description of all useful information about a heap, including a segment summary and a list of any Virtually Allocated Blocks (VABs) or extra UnCommitted Range (UCR) tables that may have been created for the heap.

The -Segment option displays a simple summary for the heap, and each of its heap-segments. Segments are created to map the linear address space for a region of a heap. A heap can be composed of up to sixteen segments.

The -Validate option is an extremely powerful option, as it completely validates a single heap-entry, or a heap and all of its components, including segments, heap-entries, and VABs. In most cases, the heap validation is equivalent to or stricter than the Win32 API Heap functions. The -Validate option is the only option that takes a heap-entry parameter as input. All other options work with heap handles or process-types. If the heap is valid, an appropriate message displays. If the validation fails, one of the following error messages appears:

- For a block whose header is corrupt:

  ```
  Generic Error: 00140BD0 is not a heap entry, or it is corrupt
  Specific Error: 00140BD0: Backward link for Block is invalid
  ```

- For a block whose guard-bytes have been overwritten:

  ```
  Allocated block: 00140BD0: Block BUSY TAIL is corrupt
  ```

  Note: If you run your application under a debugger, for example, BoundsChecker or Visual C++, each allocated block has guard-bytes, and each free block is marked with a pattern so that random overwrites can be detected.

- For a free block that has been written to, subsequent to being freed:

  ```
  Free block: 00140E50: Free block failed FREE CHECK at 141E70
  ```

Use the -Base option to change the mode in which addresses and heap entry sizes display. Under normal operation, all output shows the address of the heap-entry data, and the size of the user data for that block. When you specify the -Base option, all output shows the address of the heap-entry header, which precedes each heap-entry, and the size of the full heap-entry, including the heap-entry header and any extra data allocated for guard-bytes, or to satisfy alignment requirements. Under most circumstances you will not want to specify base addressing unless you are trying to walk a heap or its entries manually.

When you use the -Base option, the base address for each heap-entry is 8 bytes less than when -Base is not specified. This happens because the heap-entry header precedes the actual heap-entry by 8 bytes. Secondly, the size for the allocated blocks is larger because of the additional 8 bytes for the heap-entry header, guard-bytes, and, if necessary, any extra bytes needed for proper alignment. The output from the -Base option is useful for manually navigating between adjacent heap entries, or checking for memory overruns between the end of the heap-entry data and any unused space prior to the guard-bytes, which are always allocated as the last two DWORDs of the heap entry.

Note: The -Base option has no effect on input parameters. Heap-entry addresses are always assumed to be the address of the heap-entry data.

Use the -TRACE option to display the contexts of a heap trace buffer which record actions that occur within a heap. Heap trace buffers are optional and are generally not created. To enable tracing in the Win32 API, specify the HEAP_CREATE_ENABLE_TRACING flag as one of the flags to ntdll!RtlCreateHeap. You cannot use this option with

Kernel32!HeapCreate( ) because it strips out all debug-flags before calling ntdll!RtlCreateHeap. You must also be running the application under a level-3 debugger, for example, BoundsChecker or the Visual C++ debugger, so that the Win32 heap debugging options will be enabled.

Any time a process-type is passed as a parameter, any and all options are performed for each heap within the process.

The HEAP32 command and all of its options work on either a single specified heap handle or ALL the heaps for an entire process.

Example: This command performs a heap validation for all the heaps in the Test32 process:

```
HEAP 32 -v test32
```

When using bare addresses, for example, 0x140000, the current context is assumed. Use the ADDR command to change to the appropriate context.

For Not Present Memory, due to the nature of operating systems that use paging to implement virtual memory, in some cases, the actual physical memory that backs a particular linear address will not be present in memory. To be useful within this restriction, the HEAP32 command detects, avoids, and, where possible, continues to operate without the need for not present pages. In all cases where not present memory prevents the HEAP32 command from performing its work, you are notified of that condition. When possible the HEAP32 command skips not present pages and continues processing at a point where physical memory is present. Because not present memory prevents the HEAP32 command from performing a full validation of a heap, the validation routines indicate success, but let you know that only a *partial* validation could be performed.

Output

Base	Base address of the heap, that is, the heap handle.
Id	Heap ID.
Cmmt/Psnt/Rsvd	Amount of committed, present, and reserved memory used for heap entries.
Segments	Number of heap segments within the heap.
Flags	Heap flags, for example, HEAP_GROWABLE (0x02).
Process	Process that owns the heap.

If you specify the -W switch, the following information displays:

Base	This is the address of the heap entry.

Type	Type of the heap entry.

Heap Entry	Description
HEAP	Represents the heap header.
SEGMENT	Represents a heap segment.
ALLOC	Active heap entry
FREE	Inactive heap entry
VABLOCK	Virtually allocated block (VAB)

Size	Size of the heap-entry. Typically, this is the number of bytes available to the application for data storage.
Seg#	Heap segment in which the heap-entry is allocated.
Flags	Heap entry flags.

If you specify the -S switch, the following additional information displays:

Seg#	Segment number of the heap segment.
Segment Range	Linear address range that this segment maps to.
Cmmt/Psnt/Rsvd	Amount of committed, present, and reserved memory for this heap segment.
Max UCR	Maximum uncommitted range of linear memory. This value specifies the largest block that can be created within this heap segment.

Example

HEAP32

Base	Id	Cmmt/Psnt/Rsvd	Segments	Flags	Process
00230000	01	0013/0013/00ED	1	00000002	csrss
7F6F0000	02	0008/0008/00F8	1	00007008	csrss
00400000	03	001C/001A/0024	1	00004003	csrss
7F5D0000	04	0005/0005/001B	1	00006009	csrss
00460000	05	00F6/00F1/001A	2	00003002	csrss
005F0000	06	000B/000B/0005	1	00005002	csrss
7F2D0000	07	002D/002D/02D3	1	00006009	csrss
02080000	08	0003/0003/0001	1	00001062	csrss
023C0000	09	0016/0014/00EA	1	00001001	csrss

See Also

For Windows 3.1, Windows 95, and Windows NT, refer to *HEAP* on page 94.
For Windows 95, refer to *HEAP32* on page 97.

HERE

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

F7

Go to the current cursor line.

Syntax

HERE

Use

The HERE command executes until the program reaches the current cursor line. HERE is only available when the cursor is in the Code window. If the Code window is not visible or the cursor is not in the Code window, use the G command instead. Use the EC command (default key F6), if you want to move the cursor into the Code window.

To use the HERE command, place the cursor on the source statement or assembly instruction that you want to execute to. Enter HERE or press the function key that HERE is programmed to (default key F7).

The HERE command exits from SoftICE with a single, one-time execution breakpoint set. In addition, all sticky breakpoints are armed.

Execution begins at the current CS:EIP and continues until the address of the current cursor position in the Code window is encountered, the window pop-up key sequence is used, or a sticky breakpoint occurs. When SoftICE pops up, for any reason, the one-time execution breakpoint is cleared.

If the Register window is visible when SoftICE pops up, all registers that have been altered since the HERE command was issued display with the bold video attribute.

For Windows 3.1

The non-sticky execution breakpoint uses an INT 3 style breakpoint.

For Windows 95 and Windows NT

The non-sticky execution breakpoint uses debug registers unless none are available, in which case, it uses INT 3.

Example

Sets an execution breakpoint at the current cursor position, then exits from SoftICE and begins execution at the current CS:EIP.

HERE

HWND

Windows 3.1, Windows 95, Windows 98 *System Information*

Display information on Window handles.

Syntax

For Windows 3.1

HWND [*level*] [*task-name*]

For Windows 95

HWND [-x][hwnd | [[*level*][*process-name*]]

For Windows NT, refer to the HWND on page 109.

level	Windows hierarchy number. 0 is the top level, 1 is the next level and so on. The window levels represent a parent child relationship. For example, a level 1 window has a level 0 parent.
task-name	Any currently loaded Windows task. These names are available with the TASK command.
-x	Display extended information about a window.
hwnd	Windows handle.
process-name	Name of any currently loaded process.

Use

Specifying a window handle as a parameter displays only the information for that window handle. If you specify a window handle, you do not need to specify the optional parameters for level and process-name.

Output

For each window handle, the following information is displayed:

Class Name	Class name or atom of class that this window belongs to.
Window Procedure	Address of the window procedure for this window.

Example

Sample output follows for the HWND command:

HWND msword

```
Handle        hQueue         QOwner     Class         Procedure
0F4C(0)       087D           MSWORD     #32769        DESKTOP
  0FD4(1)     080D           MSWORD     #32768        MENUWND
  22C4(1)     087D           MSWORD     OpusApp       0925:0378
    53E0(2)   087D           MSWORD     OpusPmt       0945:1514
    2764(2)   087D           MSWORD     a_sdm_Msft    0F85:0010
     2800(3)  087D           MSWORD     OpusFedt      0F85:0020
     2844(3)  087D           MSWORD     OpusFedt      0F85:0020
    2428(2)   087D           MSWORD     OpusIconBar   0945:14FE
    2888(2)   087D           MSWORD     OpusFedt      0945:14D2
```

Abbreviated output follows for the HWND command:

HWND -x winword

```
Window Handle      : (0288) Level (1)
    Parent         : 16A7:000204CC
    Child          : NULL
    Next           : 16A7:00020584
    Owner          : NULL
    Window RECT    : (9,113) - (210,259)
    Client RECT    : (10,114) - (189,258)
    hQueue         : 1C97
      Size         : 16
      QOwner       : WINWORD
    hrgnUpdate     : NULL
    wndClass       : 16A7:281C
    Class          : ListBox
    hInstance      : (349E) (16 bit hInstance)
    lpfnWndProc    : 2417:000057F8
```

```
Window Handle       : (0288) Level (1)
    dwFlags1        : 40002
    dwStyle         : 44A08053
    dwExStyle       : 88
    dwFlags2        : 0
    ctrlID/hMenu    : 03E8
    WndText         : NULL
    unknown1        : 4734
    propertyList    : NULL
    lastActive      : NULL
    hSystemMenu     : NULL
    unknown2        : 0
    unknown3        : 0000
    classAtom       : C036
    unknown4        : 4CAC
    unknown5        : A0000064
```

See Also

For Windows NT, refer to *HWND* on page 109.

HWND

Windows NT *System Information*

Display information on Window handles.

Syntax

HWND [-x][-c] [***hwnd-type*** | ***desktop-type*** | ***process-type*** |
thread-type | ***module-type*** | ***class-name***]

-eXtended	Display extended information about each window handle.
-Children	Force the display of window hierarchy when searching by thread-type, module-type, or class-name.
hwnd-type	Window handle or pointer to a window structure.
desktop-type	Desktop handle or desktop pointer to a window structure (3.51 only).
process-type, thread-type or module-type	Window owner-type. A value that SoftICE can interpret as being of a specific type such as process name, thread ID, or module image base.
class name	Name of a registered window class.

Use

The HWND command enumerates and displays information about window handles.

The HWND command allows you to isolate windows that are owned by a particular process, thread or module, when you specify a parameter of the appropriate type.

For Windows 3.1 and Windows 95, refer to HWND on page 106.

The -eXtended option shows extended information about each window.

When you specify the -eXtended option, or an owner-type as a parameter, the HWND command will not automatically enumerate child windows. Specifying the -Children option forces all child windows to be enumerated (regardless of whether they meet any specified search criteria).

Output

For each HWND that is enumerated, the following information is displayed:

Handle	HWND handle (refer to *OBJTAB* on page 147 for more information). Each window handle is indented to show its child and sibling relationships to other windows.
Class	Registered class name for the window, if available (refer to *CLASS* on page 48 for more information).
WinProc	Address of the message callback procedure. Depending on the callback type, this value is displayed as a 32-bit flat address or 16-bit selector:offset.

TID	Owning thread ID.
Module	Owning module name (if available). If the module name is unknown, the module handle will be displayed as a 32-bit flat address or 16-bit selector:offset, depending on the module type.

Example

The following example uses the HWND command without parameters or options:

HWND

```
Handle     Class                            WinProc    TID   Module

01001E     #32769 (Desktop)                 5FBFE425   24    winsrv
 050060     #32770 (Dialog)                 60A29304   18    winlogon
 010044     SAS window class                022A49C4   18    winlogon
 010020     #32768 (PopupMenu)              5FBEDBD5   24    winsrv

010022     #32769 (Desktop)                 5FBFE425   24    winsrv
 010024     #32768 (PopupMenu)              5FBEDBD5   24    winsrv
 030074     Shell_TrayWnd                   0101775E   67    Explorer
  030072     Button                         01012A4E   67    Explorer
  0800AA     TrayNotifyWnd                  010216C4   67    Explorer
   03003E     TrayClockWClass               01028C85   67    Explorer
  030078     MSTaskSwWClass                 01022F69   67    Explorer
   030076     SysTabControl32               712188A8   67    Explorer
 05007A     tooltips_class32                7120B43A   67    Explorer
 03003C     tooltips_class32                7120B43A   67    Explorer
 2E00F0     NDDEAgnt                        016E18F1   4B    nddeagnt

1C0148     CLIPBOARDWNDCLASS                034F:2918  2C    OLE2
 9B0152     DdeCommonWindowClass            77C2D88B   2C    ole32
 3200F2     OleObjectRpcWindow              77C2D73B   2C    ole32
 0800A2     DdeCommonWindowClass            77C2D88B   67    ole32
 030086     OleMainThreadWndClass           77C2DCF2   67    ole32
 030088     OleObjectRpcWindow              77C2D73B   67    ole32
 03008A     ProxyTarget                     71E6869A   67    shell32
 03008C     ProxyTarget                     71E6869A   67    shell32
 030070     ProxyTarget                     71E6869A   67    shell32
 04007C     ProxyTarget                     71E6869A   67    shell32
 0400CC     OTClass                         0100D7F3   67    Explorer
 0300CA     DDEMLEvent                      5FC216AB   67    winsrv
 0300C6     DDEMLMom                        60A2779D   67    00000000
 0300C0     #42                             0BB7:0776  78    MMSYSTEM
 0300D2     WOWFaxClass                     01F9F7A8   78    WOWEXEC
 060062     ConsoleWindowClass              5FCD23C7   2B    winsrv
 0300B4     WOWExecClass                    03CF:0B3E  78    WOWEXEC
```

```
 030068     Progman                      0101B1D3   67   Explorer
  0E00BC     SHELLDLL_DefView            71E300E8   67   shell32
   040082     SysListView32              7121A0EC   67   shell32
    030080     SysHeader32               7120B06F   67   shell32
```

Notes: You may have noticed that the output from the previous example enumerated two desktop windows (handles 1001E and 10022), each with its own separate window hierarchy. This is because the system can create more than one object of type Desktop, and each Desktop object has its own Desktop Window which defines the window hierarchy. If you use the HWND command in a context that does not have an assigned Desktop, the HWND command enumerates all objects of type Desktop.

Because the system may have create more than one object of type Desktop, the HWND command accepts a Desktop-type handle as a parameter. This allows the window hierarchy for a specific Desktop to be enumerated. You can use the command OBJTAB DESK to enumerate all existing desktops in the system.

The following is an example of using the HWND command for a specific window handle:

HWND 400a0

```
Handle     Class                        WinProc    TID  Module
0400A0     Progman                      0101B1D3   74   Explorer
```

The following is an example of enumerating only those windows owned by thread 74:

HWND 74

```
Handle     Class                        WinProc       TID  Module
 2F00F0     Shell_TrayWnd               0101775E      74   Explorer
  0500CE     Button                     01012A4E      74   Explorer
  0500C4     TrayNotifyWnd              010216C4      74   Explorer
   040074     TrayClockWClass           01028C85      74   Explorer
  0500C6     MSTaskSwWClass             01022F69      74   Explorer
   0400C8     SysTabControl32           712188A8      74   Explorer
 3700F2     tooltips_class32            7120B43A      74   Explorer
 040066     tooltips_class32            7120B43A      74   Explorer
 0F00BC     DdeCommonWindowClass        77C2D88B      74   ole32
 040068     OleMainThreadWndClass       77C2DCF2      74   ole32
 0500CC     OleObjectRpcWindow          77C2D73B      74   ole32
 2600BA     ProxyTarget                 71E6869A      74   shell32
 0400D0     ProxyTarget                 71E6869A      74   shell32
 0400CA     ProxyTarget                 71E6869A      74   shell32
 070094     ProxyTarget                 71E6869A      74   shell32
 04009E     OTClass                     0100D7F3      74   Explorer
 480092     DDEMLEvent                  5FC216AB      74   winsrv
 09004A     DDEMLMom                    60A2779D      74   00000000
```

```
 0400A0     Progman                     0101B1D3     74   Explorer
  0500C0     SHELLDLL_DefView           71E300E8     74   shell32
   070090     SysListView32             7121A0EC     74   shell32
    050096     SysHeader32              7120B06F     74   shell32
```

Note: A process-name always overrides a module of the same name. To search by module, when there is a name conflict, use the module handle (base address or module-database selector) instead. Also, module names are always context sensitive. If the module is not loaded in the current context (or the CSRSS context), the HWND command interprets the module name as a class name instead.

The following example shows the output when the -eXtended option is used:

HWND -x 400a0

```
Hwnd           : 0400A0    (7F2D7148)
Class Name     : Progman
Module         : Explorer
Window Proc    : 0101B1D3
Win Version    : 4.00
Title          : Program Manager
Desktop        : 02001F    (00402D58)
Parent         : 010022    (7F2D0C28)
1st Child      : 0500C0    (7F2D7600)
Style          : CLIPCHILDREN | CLIPSIBLINGS | VISIBLE | POPUP
Ex. Style      : TOOLWINDOW | A0000000
Window Rect    : 0, 0, 1024, 768 (1024 x 768)
Client Rect    : 0, 0, 1024, 768 (1024 x 768)
```

See Also

For Windows 3.1 and Windows 95, refer to *HWND* on page 106.

I

Windows 3.1, Windows 95, Windows 98, Windows NT *I/O Port*

Input a value from an I/O port.

Syntax

I[*size*] *port*

size

Value	Description
B	Byte
W	Word
D	DWORD

port Port address.

Use

The I command in most cases does an actual I/O instruction so it is showing the actual state of the hardware port. In the case of virtualized ports, the actual data may not be the same as the virtualized data that an application would see.

The only ports that SoftICE does not do I/O on are the interrupt mask registers (Port 21 and A1). For those ports, SoftICE shows the value that existed when SoftICE popped up.

Use the input from port commands to read and display a value from a hardware port. Input can be done from byte, word, or dword ports. If you do not specify size, the default is B.

Example

Performs an input from port 21, which is the mask register for interrupt controller one.

```
I 21
```

I1HERE

Windows 3.1, Windows 95, Windows 98, Windows NT *Mode Control*

Pop up on embedded INT 1 instructions.

Syntax

I1HERE [**on** | **off**]

Use

Use the I1HERE command to specify that any embedded interrupt 1 bring up the SoftICE screen. This feature is useful for stopping your program in a specific location. Before popping up, SoftICE checks to see that there is really an INT 1 in the code. If there is not, SoftICE will not pop up.

To use this feature, place an INT 1 into the code immediately before the location where you want to stop. When the INT 1 occurs, it brings up the SoftICE screen. At this point, the current EIP is the instruction after the INT 1 instruction.

If you do not specify a parameter, the current state of I1HERE displays.

The default is I1HERE off.

This command is useful when you are using an application debugging tool such as BoundsChecker. Since these tools rely on INT 3's for breakpoint notifications, you should use INT 1s in your code so that the tools do not become confused when your hardwired interrupts occur.

For Windows 3.1 and Windows 95

VMM, the Windows memory management VxD, executes INT 1 instructions prior to certain fatal exits. If you have I1HERE ON, you can trap these. The INT 1s generated by VMM are most often caused by a page fault with the registers set up as follows:

- EAX=faulting address
- ESI points to an ASCII message
- EBP points to a CRS (Client Register Structure as defined in the DDK include file VMM.INC).

Example

Turns on I1HERE mode. Any INT 1s generated after this point bring up the SoftICE screen.

```
I1HERE on
```

I3HERE

Windows 3.1, Windows 95, Windows 98, Windows NT *Mode Control*

Pop up on INT 3 instructions.

Syntax

I3HERE [on | off]

Use

Use the I3HERE command to specify that any interrupt 3 pop up SoftICE. This feature is useful for stopping your program in a specific location.

To use this feature, place an INT 3 into your code immediately before the location where you want to stop. When the INT 3 occurs, it brings up the SoftICE screen. At this point, the current EIP is the instruction after the INT 3 instruction.

If you are developing a Windows program, the DebugBreak( ) Windows API routine performs an INT 3.

If you do not specify a parameter, the current state of I3HERE displays.

Note: If you are using an application debugging tool such as the Visual C debugger or NuMega's BoundsChecker, you should place INT 1s in your code instead of INT 3s. Refer to *I1HERE* on page 114.

Example

Turns on I3HERE mode. Any INT 3s generated after this point cause SoftICE to pop up.

```
I3HERE on
```

When the command I3HERE==ON, and you are using a level -3 debugger, such as BoundsChecker, SoftICE traps on any INT 3 breakpoints installed by the level-3 debugger. When this happens, set I3HERE==OFF, and use the GENINT command to reactivate the breakpoint. This returns control to the level -3 debugger, and SoftICE does not trap further INT 3s.

```
I3HERE off
GENINT 3
```

See Also

GENINT, I3HERE, SET

IDT

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the Interrupt Descriptor Table.

Syntax

IDT [***interrupt-number***]

interrupt-number Interrupt-number to display information

Use

The IDT command displays the contents of the Interrupt Descriptor Table after reading the IDT register to obtain its address.

The IDT command without parameters displays the IDT's base address and limit, as well as the contents of all entries in the table. If you specify an optional interrupt-number, only information about that entry is displayed.

For Windows NT

Almost all interrupt handlers reside in NTOSKRNL, so it is very useful to have exports loaded for it so that the handler names are displayed.

Note: NTOSKRNL must be the current symbol table (refer to *TABLE* on page 194) to view symbol names.

Output

Each line of the display contains the following information:

interrupt number 0 - 0FFh (5Fh for Windows 3.1, Windows 95).

interrupt type One of the following:

Type	Description
CallG32	32-bit Call Gate
CallG16	16-bit Call Gate
TaskG	Task Gate
TrapG16	16-bit Trap Gate
TrapG32	32-bit Trap Gate
IntG32	32-bit Interrupt Gate
IntG16	16-bit Interrupt Gate

address Selector:offset of the interrupt handler.

selector's DPL	Selector's descriptor privilege level (DPL), which is either 0, 1, 2 or 3.
present bit	P or NP, indicating whether the entry is present or not present.
Owner+Offset	For Windows 95 and Windows NT only: Symbol or owner name plus the offset from that symbol or owner.

Example

The following command shows partial output of the IDT command with no parameters:

:IDT

```
Int   Type     Sel:Offset     Attributes Symbol/Owner
IDTbase=C000ABBC  Limit=02FF
0000  IntG32   0028:C0001200  DPL=0  P   VMM(01)+0200
0001  IntG32   0028:C0001210  DPL=3  P   VMM(01)+0210
0002  IntG32   0028:C00EEDFC  DPL=0  P   VTBS(01)+1D04
0003  IntG32   0028:C0001220  DPL=3  P   VMM(01)+0220
0004  IntG32   0028:C0001230  DPL=3  P   VMM(01)+0230
0005  IntG32   0028:C0001240  DPL=3  P   VMM(01)+0240
0006  IntG32   0028:C0001250  DPL=0  P   VMM(01)+0250
0007  IntG32   0028:C0001260  DPL=0  P   VMM(01)+0260
0008  TaskG    0068:00000000  DPL=0  P
0009  IntG32   0028:C000126C  DPL=0  P   VMM(01)+026C
000A  IntG32   0028:C000128C  DPL=0  P   VMM(01)+028C
```

The next command shows the contents of one entry in the IDT:

:IDT d

```
Int   Type     Sel:Offset     Attributes Symbol/Owner
000D  IntG32   0028:C00012B0  DPL=0  P   VMM(01)+02B0
```

IRP

Windows NT *System Information*

Display information about an I/O Request Packet (IRP).

Syntax

IRP [***irp-address***]

irp-address Address of the start of the IRP structure to be displayed.

Use

The IRP command displays the contents of the I/O Request Packet and the contents of associated current I/O stack located at the specified address. The command does not check for the validity of the IRP structure being shown, so any address will be accepted by SoftICE as an irp-address.

The IRP fields shown by SoftICE are not documented in their entirety here, as adequate information about them can be found in NTDDK.H in the Windows NT DDK. A few fields deserve special mention, however, since device driver writers find them particularly useful:

Flags Flags used to define IRP attributes.

StackCount The number of stack locations that have been allocated for the IRP. A common device driver bug is to access non-existent stack locations, so this value may be useful in determining when this has occurred.

CurrentLocation This number indicates which stack location is the current one for the IRP. Again, this value, combined with the previous StackCount, can be used to track down IRP stack-related bugs.

Cancel This boolean is set to TRUE if the IRP has been cancelled as a result of an IRP cancellation call. This happens when the IRP's result is no longer needed so the IRP will not complete.

Tail.Overlay.
CurrentStackLoc Address of current stack location. The contents of this stack location are displayed after the IRP, as illustrated in the example for this command.

Cancel This boolean is set to TRUE if the IRP has been cancelled as a result of an IRP cancellation call. This happens when the IRP's result is no longer needed so the IRP will not complete.

These fields in the current stack location may be useful:

Major Function and Minor Function — These fields indicate what type of request the IRP is being used for. The major function is used in determining which request handler will be called when an IRP is received by a device driver.

Device Object — Pointer to the device object that the IRP is currently stationed at. In other words, the IRP has been sent to, and is in the process of being received by, the device driver owning the device object.

File Object — Pointer to the file object associated with the IRP. It can contain additional information that serves as IRP parameters. For example, file system drivers use the file object path name field to determine the target file of a request.

Completion Rout — This field is set when a driver sets a completion routine for an IRP through the IoSetCompletionRoutine call. Its value is the address of the routine that will be called when a lower-level driver (associated with a stack location one greater than the current one) completes servicing of the IRP and signals that it has done so with IoCompleteRequest.

Example

The following example shows the output for the IRP command:

:IRP eax

```
MdlAddress *       : 00000000
Flags              : 00000404 IRP_SYNCHRONOUS_API|IRP_CLOSE_OPERATION
AssociatedIrp      : 00000000
&ThreadListEntry   : FD8D9B18
IoStatus           : 00000000
RequestorMode      : 00
PendingReturned    : False
StackCount         : 03
CurrentLocation    : 03
Cancel             : False
CancelIrql         : 00
ApcEnvironment     : 00
Zoned              : True
UserIosb *         : FD8D9B20
UserEvent *        : FB11FB40
Overlay            : 00000000 00000000
CancelRoutine *    : 00000000
UserBuffer *       : 00000000
Tail.Overlay
    &DeviceQueueEntry : FD8D9B48
    Thread *          : FD80A020
    AuxiliaryBuffer * : 00000000
```

```
    &ListEntry        : FD8D9B60
    CurrentStackLoc * : FD8D9BC0
    OrigFileObject *  : FD819E08
Tail.Apc *        : FD8D9B48
Tail.ComplKey     : 00000000
CurrentStackLocation:
MajorFunction     : 12 IRP_MJ_CLEANUP
MinorFunction     : 00
Control           : 00
Flags             : 00
Others            : 00000000 00000000 00000000 00000000
DeviceObject *    : FD851E40
FileObject *      : FD819E08
CompletionRout *  : 00000000
Context *         : 00000000
```

LDT

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the Local Descriptor Table.

Syntax

LDT [***selector***]

selector	Starting LDT selector to display.

Use

The LDT command displays the contents of the Local Descriptor Table after reading its location from the LDT register. If there is no LDT, an error message will be printed. If you specify an optional selector, only information on that selector is displayed. If the starting selector is a GDT selector (bit 2 is 0), the GDT displays rather than the LDT. The first line of output contains the base address and limit of the LDT.

For Windows 95 and Windows NT

Even when there is no LDT, the LDT command can display an LDT you supply as a command parameter. This optional parameter can be a GDT selector that represents an LDT. You can locate selectors of type LDT with the GDT command.

For Windows NT

The LDT command is process specific and only works in processes that have an LDT. Use the ADDR command to determine which processes contain LDTs. Use ADDR to switch to those processes, then use the LDT command to examine their LDTs.

Output

Each line of the display contains the following information:

selector value Lower two bits of this value reflect the descriptor privilege level.

selector type

Type	Description
Code16	16-bit code selector
Data16	16-bit data selector
Code32	32-bit code selector
Data32	32-bit data selector
CallG32	32-bit Call Gate selector
CallG16	16-bit Call Gate selector

Type	Description
TaskG32	32-bit Task Gate selector
TaskG16	16-bit Task Gate selector
TrapG32	32-bit Trap Gate selector
TrapG16	16-bit Trap Gate selector
IntG32	32-bit Interrupt Gate selector
IntG16	16-bit Interrupt Gate selector
Reserved	Reserved selector

selector base Linear base address of the selector.

selector limit Size of the selector.

selector DPL Selector's descriptor privilege level (DPL), either 0, 1, 2 or 3.

present bit P or NP, indicating whether the selector is present or not present.

segment attributes One of the following:

Type	Description
RW	Data selector is readable and writable.
RO	Data selector is read only.
RE	Code selector is readable and executable.
EO	Code selector is execute only.
B	TSS's busy bit is set.

Example

The following example shows sample output for the LDT command.

:LDT

```
Sel.  Type      Base      Limit     DPL  Attributes
LDTbase=8008B000  Limit=4FFF
0004  Reserved  00000000  00000000  0    NP
000C  Reserved  00000000  00000000  0    NP
0087  Data16    80001000  00000FFF  3    P   RW
008F  Data16    00847000  0000FFFF  3    P   RW
0097  Data16    0002DA80  0000021F  3    P   RW
009F  Data16    00099940  000029FF  3    P   RW
00A7  Data16    0001BAC0  000000FF  3    P   RW
00AF  Data16    C11D9040  0000057F  3    P   RW
```

LHEAP

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the Windows local heap.

Syntax

LHEAP [***selector*** | ***module-name***]

selector	LDT data selector.
module-name	Name of any 16-bit module.

Use

The LHEAP command displays the data objects that a Windows program has allocated on the local heap. If you do not specify a selector, the value of the current DS register is used. The specified selector is usually the Windows program's data selector. To find this, use the HEAP command on the Windows program you are interested in and look for an entry of type data. Each selector that contains a local heap is marked with the tag LH.

If a module-name is entered, SoftICE uses the modules default data segment for the heap walk.

For Windows 95 and Windows NT

To find all segments that contain a local heap, use the HEAP command with the -L option.

For Windows NT

The LHEAP command only works if the current process contains a WOW box.

Output

For each local heap entry the following information displays:

offset	16-bit offset relative to the specified selector base address.
size	Size of the heap entry in bytes.
type	Type of entry. One of the following:

Type	Description
FIX	Fixed (not moveable)
MOV	Moveable
FREE	Available memory

handle Handle associated with each element. For fixed elements, the handle is equal to the address that is returned from LocalAlloc( ). For moveable elements, the handle is the address that will be passed to LocalLock( ).

At the end of the list, the total amount of memory in the local heap displays.

Example

To display all local heap entries belonging to the GDI default local heap, use the following command:

`LHEAP gdi`

```
Offset    Size      Type      Handle
93D2      0046      Mov       0DFA
941E      0046      Mov       0C52
946A      0046      Mov       40DA
94B6      004E      Mov       0C66
950A      4A52      Mov       0E52
Used: 19.3K
```

LINES

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Change the number of lines for the SoftICE display.

Syntax

For Windows 3.1

LINES [25 | 43 | 50]

For Windows 95 and Windows NT

With Universal Video Driver:

LINES *numlines*

numlines	Number of screen lines. Set this to any value greater than 25.

With VGA Text Video Driver:

LINES [25 | 43 | 50 | 60]

Use

The LINES command changes SoftICE's character display mode. For VGA Text Driver displays, it allows different display modes: 25-line, 43-line, 50-line, and 60-line mode. The 43-, 50-, and 60-line modes are only valid on VGA display adapters. For the Universal Video Driver, you can specify any number of lines greater than 25.

Using LINES with no parameters displays the current state of LINES. The default number of display lines is 25.

If you enter the ALTSCR command, SoftICE changes to 25-line mode automatically. If you change back to a VGA display and want a larger line mode, enter the LINES command again. To display in 50-line mode on a serial terminal, first place the console mode of the serial terminal into 50-line mode using the DOS MODE command.

For Windows 95 and Windows NT

You can display 60 lines for single monitor debugging.

When debugging in serial mode, all line counts are supported for VGA displays.

Example

To change the SoftICE display to 53 lines using the Universal Video Driver, use the following command. The current font affects the number of lines SoftICE can display.

```
LINES 53
```

See Also

SET, WIDTH

LOCALS

Windows 95, Windows 98, Windows NT *Symbol/Source Command*

Lists local variables from the current stack frame.

Syntax

LOCALS

Use

Use the LOCALS command to list local variables from the current stack frame to the Command window.

Output

The following information displays for each local symbol:

- Stack Offset
- Type definition
- Value, Data, or structure symbol ({...})

The type of local determines whether a value, data, or structure symbol ({...}) is displayed. If the local is a pointer, the data it points to is displayed. If it is a structure, the structure symbol is displayed. If the local is neither a pointer nor a structure, its value is displayed.

Hint: You can expand structures, arrays, and character strings to display their contents. Use the WL command to display the Locals window, then double-click the item you want to expand. Note that expandable items are delineated with a plus (+) mark.

Example

The following example displays the local variables for the current stack frame:

:LOCALS

```
[EBP-4] struct_BOUNCEDATA * pdb=0x0000013F <{...}>
[EBP+8] void * hWnd=0x000006D8
```

See Also

TYPES, WL

M

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Move data.

Syntax

`M` ***`source-address`*** **`l`** ***`length dest-address`***

source-address	Start of address range to move.
length	Length in bytes.
dest-address	Start of destination address range.

Use

The specified number of bytes are moved from the source-address to the dest-address.

Example

Moves 2000h bytes (8KB) from memory location DS:1000h to ES:5000h.

```
M ds:1000 l 2000 es:5000
```

MACRO

Windows 95, Windows 98, Windows NT *Customization*

Define a new command that is a superset of SoftICE commands.

Syntax

MACRO [***macro-name***] | [*****] | [**= *"macro body"***]

macro-name	Case-insensitive, 3-8 character name for the macro being defined, or the name of an existing macro.
macro-body	Quoted string that contains a list of SoftICE commands and parameters separated by semi-colons (;).
*	Delete one or all defined macros.
=	Define (or redefine) a macro.

Use

The MACRO command is used to define new Macro commands that are supersets of existing SoftICE commands. Defined macros can be executed directly from the SoftICE command line. The MACRO command is also used to list, edit, or delete individual macros. Macros are directly related to breakpoint actions, as breakpoint actions are simply macros that do not have names, and can only be executed by the SoftICE breakpoint engine.

If no options are provided, a list of all defined macros will be displayed, or if a macro-name is specified, that macro will be inserted into the command buffer so that it can be edited.

When defining or redefining a macro, the following form of the macro command is used:

MACRO *macro-name = "macro-body"*

The macro-name parameter can be between 3 and 8 characters long, and may contain any alphanumeric character or underscore (_). If the macro-name parameter specifies an existing macro, the existing macro will be redefined. The macro-name cannot be a duplicate of an existing SoftICE command. The macro-name must be followed by an equal sign "=", which must be followed by the quoted string that defines the macro-body.

The macro-body parameter must be embedded between beginning and ending quotation marks ("). The macro-body is made up of a collection of existing SoftICE commands, or defined macros, separated by semi-colons. Each command may contain appropriate 'literal' parameters, or can use the form%<parameter#>, where parameter# must be between 1 and 8. When the macro is executed from the command line, any parameter references will expand into the macro-body from the parameters specified when the command was executed. If you need to embed a literal quote character (") or a percent sign (%) within the macro body precede the character with a backslash character (\). Because the backslash character is used for escape sequences, to specify a literal backslash character, use two consecutive backslashes (\\). The final command within the macro-body does not need to be terminated by a semi-colon.

You can define macros in the SoftICE Loader using the same syntax described here. When you load SoftICE, each macro definition is created and available for use. SoftICE displays a message for each defined macro to remind you of it presence. Since macros consume memory, you can set the maximum number of named and unnamed macros (that is, breakpoint actions) that can be defined during a SoftICE session. The default value of 32 is also the minimum value. The maximum value is 256.

Note: A macro-body cannot be empty. It must contain one or more non-white space characters. A macro-body can execute other macros, or define another macro, or even a breakpoint with a breakpoint action. A macro can even refer to itself, although recursion of macros is not extremely useful because there is no programmatic way to terminate the macro. Macros that use recursion execute up to the number of times that SoftICE permits (32 levels of recursion are supported), no more, and no less. Even with this limitation, macro recursion, although crude, can be useful for walking nested or linked data structures. To get a recursive macro to execute as you expect, you have to devise clever macro definitions.

Example

The following is an example of using the MACRO command without parameters or options:

:MACRO

```
XWHAT   = "WHAT EAX;WHAT EBX;WHAT ECX; WHAT EDX; WHAT ESI; WHAT EDI"
OOPS    = "I3HERE OFF;GENINT 3"
1shot   = "bpx eip do \"bc bpindex \""
```

Note: The name of the macro is listed to the left, and the macro body definition to the right.

The following are more examples of basic usage of the MACRO command:

:MACRO *	Delete all named macros.
:MACRO oops *	Delete the macro named oops.
:MACRO xwhat	Edit the macro named xwhat.

Note: Because macros can be redefined at any time, when you use the edit form of the MACRO command (MACRO *macro-name*) the macro definition will be placed in the edit buffer so that it can be edited. If you do not wish to modify the macro, press ESC. The existing macro will remain unchanged. If you modify the macro-body without changing the macro name, the macro will be redefined (assuming the syntax is correct!)

The following is a simple example of a macro definition:

:MACRO help = “h”

The next example uses a literal parameter within the macro-body. Its usefulness is limited to specific situations or values:

:MACRO help = "h exp"

In this example, the SoftICE H command is executed with the parameter EXP every time the macro executes. This causes the help for the SoftICE EXP command to display.

This is a slightly more useful definition of the same macro:

:MACRO help= "help %1"

In this example, an optional parameter was defined to pass to the SoftICE H command. If the command is executed with no parameters, the argument to the H command is empty, and the macro performs exactly as the first definition; help for all commands is displayed. If the macro executes with 1 parameter, the parameter is passed to the H command, and the help for the command specified by parameter 1 is displayed. For execution of macros, all parameters are considered optional, and any unused parameters are ignored.

The following are examples of legal macro definitions:

:MACRO qexp = "addr explorer; query %1" qexp

or

qexp 1 40000

:MACRO 1shot = "bpx %1 do \"bc bpindex\"" 1shot eip

or

1shot @esp

:MACRO ddt = "dd thread" ddt

:MACRO ddp = "dd process" ddp

:MACRO thr = "thread %1 tid" thr

or

thr -x

The following are examples of *illegal* macro definitions, with an explanation and a corrected example:

Illegal Definition: ~~MACRO dd = "dd dataaddr"~~
Explanation: This is a duplication of a SoftICE command. SoftICE commands cannot be redefined.
Corrected Example: MACRO dda = "dd dataaddr"

Illegal Definition: ~~`MACRO aa = “addr %1”`~~
Explanation: The macro command name is too short. A macro name must be between 3 and 8 characters long.
Corrected Example: `MACRO aaa = “addr %1”`

Illegal Definition: ~~`MACRO pbsz = ? hibyte(hiword(*(%1-8))) << 5`~~
Explanation: The macro body must be surrounded by quote characters (“).
Corrected Example: `MACRO pbsz = “? hibyte(hiword(*(%1-8))) << 5”`

Illegal Definition: ~~`MACRO tag = “? *(%2-4)”`~~
Explanation: The macro body references parameter %2 without referencing parameter %1. You cannot reference parameter %n+1 without having referenced parameter %n.
Corrected Example: `MACRO tag = “? *(%1-4)”`

MAP32

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display a memory map of all 32-bit modules currently loaded in memory.

Syntax

For Windows 3.1

MAP32 [***module-name*** | ***module-handle***]

module-name	Windows module-name.
module-handle	Base address of a module image.

For Windows 95 and Windows NT

MAP32 [***module-name*** | ***module-handle*** |***address***]

module name	Windows module-name.
module handle	Base address of a module image.
address	Any address that falls within an executable image.

Use

MAP32 with no parameters lists information about all 32-bit modules.

If you specify either a module-name or module-handle as a parameter, only sections from the module are shown. For each module, one line of data is printed for every section belonging to the module.

Since the MAP32 command takes any address that falls within an executable image, an easy way to see the memory map of the module that contains the current EIP is to enter:

MAP32 eip

For Windows 95

No matter what process/context you are in, you see the same list of drivers because memory above 2GB is globally mapped.

You see different lists of applications/DLLs because they are *always* private to an address context.

For Windows NT

MAP32 lists kernel drivers as well as applications and DLLs that exist in the current process. They can be distinguished in the map because drivers always occupy addresses above 2GB, while applications and DLLs are always below 2GB.

Output

Each line in MAP32's output contains the following information:

Owner	Module name.
Name	Section name from the executable file.
Obj#	Section number from the executable file.
Address	Selector:offset address of the section.
Size	Section's size in bytes.
Type	Type and attributes of the section, as follows:

Type	Attributes
CODE	Code
IDATA	Initialized Data
UDATA	Uninitialized Data
RO	Read Only
RW	Read/Write
SHARED	Object is shared

Example

For Windows 3.1

The following example illustrates sample output for MAP32 executed on a Visual C module.

:MAP32 msvcrt10

```
Owner     Obj Name  Obj#     Address        Size      Type
MSVCRT10  .text     0001     2197:86C81000  00024A00  CODE RO
MSVCRT10  .bss      0002     219F:86CA6000  00001A00  UDATA RW
MSVCRT10  .rdata    0003     219F:86CA8000  00000200  IDATA RO
```

```
MSVCRT10  .edata   0004     219F:86CA9000  00005C00  IDATA RO
MSVCRT10  .data    0005     219F:86CAF000  00006A00  IDATA RW
MSVCRT10  .idata   0006     219F:86CB6000  00000A00  IDATA RW
MSVCRT10  .reloc   0007     219F:86CB7000  00001800  IDATA RO
```

MAPV86

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the DOS memory map of the current Virtual Machine.

Syntax

MAPV86 [***address***]

address	Segment:offset type address.

Use

If no address parameter is specified, a map of the entire current virtual machine's V86 address space is displayed. Information about the area in the map where a certain address lies can be obtained by specifying the address.

Pages of DOS VM memory may not be valid (not mapped in) when you enter the MAPV86 command. If this occurs, the output from the MAPV86 command will terminate with a PAGE NOT PRESENT message. Often, just popping out of, and then back into, SoftICE will result in those pages being mapped in.

A useful application of the MAPV86 command is in obtaining addresses to which a symbol table must be aligned with the SYMLOC command. DOS programs that were started before Windows will not automatically have their symbol information mapped to their location in V86 memory. By obtaining the start of their static code segment (and adding 10h to it if the program is a .EXE) with the MAPV6 command, and setting the symbol table alignment to that value, source level debugging for these global DOS programs is possible.

For Windows NT

The MAPV86 command is process specific. You must be in an NTVDM process because these are the only ones that contain V86 boxes. There is no *global* MSDOS in Windows NT.

Output

For Windows 3.1 and Windows 95

The following summary information is displayed by the MAPV86 command:

VM ID	Virtual machine (VM) ID. VM1 is the System VM.
VM handle	32-bit virtual machine handle.
CRS pointer	VM's 32-bit client register structure pointer.
VM address	32-bit linear address of the VM. This is the *high linear* address of the virtual machine, which is also currently mapped to linear address 0.

If the current CS:IP belongs to a MAPV86 entry, that line will be highlighted. Each line of the MAPV86 display contains the following information:

Start	Segment:offset start address of the component.
Length	Length of the component in paragraphs.
Name	Owner name of the component.

Example

The following example illustrates how to use the MAPV86 command to display the entire V86 map for the current VM:

`:MAPV86`

`ID=01 Handle=80441000 CRS Ptr=80013390 Linear=80C00000`

Start	Length	Name
0000:0000	0040	Interrupt Vector Table
0040:0000	0030	ROM BIOS Variables
0070:0000	025D	I/O System
02CD:0000	08E6	DOS
0BB5:0012	0000	NUMEGA
0C8B:0000	00E8	SOFTICE1
0D41:0000	00B6	XMSXXXX0
10D0:0000	038F	SMARTAAR

MOD

Windows 3.1 *System Information*

Display the Windows module list.

Syntax

MOD [***partial-name***]

partial-name	Prefix of the Windows module name.

Use

This command displays the Windows module list in the Command window. A module is a Windows application or DLL. All 16-bit modules will be displayed first, followed by all 32 bit modules. If a partial name is specified, only those modules that begin with the name will be displayed.

Output

For each loaded module the following information is displayed:

For Windows 95 and Windows NT, refer to MOD on page 139.

module handle	16-bit handle that Windows assigns to each module. It is actually a 16-bit selector of the module database record which is similar in format to the EXE header of the module file.
pe-header	Selector:offset of the PE File header for that module. *Note:* A value will only be displayed in this column for 32-bit modules.
module name	Name specified in the .DEF file using the 'NAME' or 'LIBRARY' keyword.
file name	Full path and file name of the module's executable file.

Example

The following example shows abbreviated output of MOD to display all modules in the system:

```
:MOD

hMod  PEHeader        Module Name    .EXE File Name
0117                  KERNEL         C:\WINDOWS\SYSTEM\KRNL386.EXE
0147                  SYSTEM         C:\WINDOWS\SYSTEM\SYSTEM.DRV
014F                  KEYBOARD       C:\WINDOWS\SYSTEM\KEYBOARD.DRV
0167                  MOUSE          C:\WINDOWS\SYSTEM\LMOUSE.DRV
01C7                  DISPLAY        C:\WINDOWS\SYSTEM\VGA.DRV
01E7                  SOUND          C:\WINDOWS\SYSTEM\MMSOUND.DRV
```

```
hMod  PEHeader       Module Name    .EXE File Name
0237                 COMM           C:\WINDOWS\SYSTEM\COMM.DRV
0000  2987:80756080  W32SKRNL       C:\WINDOWS\SYSTEM\win32s\w32skrnl.dll
12C7  2987:86C20080  FREECELL       C:\WIN32APP\FREECELL\FREECELL.EXE
1FC7  2987:86C40080  CARDS          C:\WIN32APP\FREECELL\CARDS.dll
1FDF  2987:86C70080  w32scomb       C:\WINDOWS\SYSTEM\win32s\w32scomb.dll
```

See Also

For Windows 95 and Windows NT, refer to *MOD* on page 139.

MOD

Windows 95 , Windows 98, Windows NT *System Information*

Display the Windows module list.

Syntax

`MOD` `[`***`partial-name`***`]`

partial-name	Prefix of the Windows module name

Use

This command displays the Windows module list in the Command window. If a partial name is specified, only modules that begin with the name will be displayed. SoftICE displays modules in the following order:

For Windows 3.1, refer to MOD on page 137.

- 16-bit modules
- 32-bit driver modules (Windows NT only)
- 32-bit application modules

For Windows 95

The module list is global. A module is a Windows application or DLL. All modules have an hMod value.

For Windows NT

The Mod command is process specific. All modules will be displayed that are visible within the current process. This includes all 16-bit modules, all 32-bit modules, and all driver modules. This means if you want to see specific modules, you must switch to the appropriate address context before using the MOD command.

You can distinguish application modules from driver modules because application modules have base addresses below 2GB (80000000h).

The 16-bit modules will be the only modules that have an hMod value.

Output

For each loaded module the following information is displayed:

module handle	16-bit handle that Windows assigns to each module. It is actually a 16-bit selector of the module database record which is similar in format to the EXE header of the module file.
base	Base linear address of the executable file. This is also used as the module handle for 32-bit executables. *Note:* A value will only be displayed in this column for 32-bit modules.

pe-header	Selector:offset of the PE File header for that module. *Note:* A value will only be displayed in this column for 32-bit modules.
module name	Name specified in the .DEF file using the 'NAME' or 'LIBRARY' keyword.
file name	Full path and file name of the module's executable file.

Example

The following example is abbreviated output of MOD used on the NTVDM WOW process:

:MOD

hMod	Base	PEHeader	ModuleName	File Name
021F			KERNEL	D:\WINNT35\SYSTEM32\KRNL386.EXE
020F			SYSTEM	D:\WINNT35\SYSTEM32\SYSTEM.DRV
01B7			KEYBOARD	D:\WINNT35\SYSTEM32\KEYBOARD.DRV
02B7			MOUSE	D:\WINNT35\SYSTEM32\MOUSE.DRV
02CF			DISPLAY	D:\WINNT35\SYSTEM32\VGA.DRV
02E7			SOUND	D:\WINNT35\SYSTEM32\SOUND.DRV
0307			COMM	D:\WINNT35\SYSTEM32\COMM.DRV
031F			USER	D:\WINNT35\SYSTEM32\USER.EXE
0397			GDI	D:\WINNT35\SYSTEM32\GDI.EXE
0347			WOWEXEC	D:\WINNT35\SYSTEM32\WOWEXEC.EXE
03DF			SHELL	D:\WINNT35\SYSTEM32\SHELL.DLL
0C3F			WFWNET	D:\WINNT35\SYSTEM32\WFWNET.DRV
0BFF			MMSYSTEM	D:\WINNT35\SYSTEM32\MMSYSTEM.DLL
0BF7			TIMER	D:\WINNT35\SYSTEM32\TIMER.DRV
	80100000	80100080	ntoskrnl	\WINNT35\System32\ntoskrnl.exe
	80400000	80400080	hal	\WINNT35\System32\hal.dll
	80010000	80010080	atapi	atapi.sys
	80013000	80013080	SCSIPORT	\WINNT35\System32\Drivers\SCSIPORT.SYS
	80001000	80001080	Atdisk	Atdisk.sys

hMod	Base	PEHeader	ModuleName	File Name
	8001B000 8001B080		Scsidisk	Scsidisk.sys
	803AE000 803AE080		Fastfat	Fastfat.sys
	FB000000 FB000080		Floppy	\SystemRoot\System32\Drivers\Floppy.SYS
	FB010000 FB010080		Scsicdrm	\SystemRoot\System32\Drivers\Scsicdrm.SYS
	FB020000 FB020080		Fs_Rec	\SystemRoot\System32\Drivers\Fs_Rec.SYS
	FB030000 FB030080		Null	\SystemRoot\System32\Drivers\Null.SYS

See Also

For Windows 3.1, refer to *MOD* on page 137.

NTCALL

Windows NT *System Information*

Display NTOSKRNL calls used by NTDLL.

Syntax

NTCALL

Use

The NTCALL command displays all NTOSKRNL calls that are used by NTDLL. Many of the API's in NTDLL are nothing more than a wrapper for routines in NTOSKRNL, where the real work is done at level 0. If you use SoftICE to step through one of these calls, you will see that it immediately performs an INT 2Eh instruction. The INT 2Eh instructions serve as the interface for transitions between a privilege level 3 API and a privilege level 0 routine that actually implements the call.

When an INT 2Eh is executed, the EDX register is set to point at the parameter stack frame for the API and the EAX register is set to the index number of the function. When the current instruction pointer reference is an INT 2Eh instruction, the SoftICE disassembler will show the address of the privilege level 0 routine that will be called when the INT 2Eh executes, along with the number of dword parameters that are being passed in the stack frame pointed at by EDX. If you wish to see the symbol name of the routine, you must load symbols for NTOSKRNL and make sure that it is the current symbol table. Refer to *TABLE* on page 194.

Output

The NTCALL command display all the level 0 API's available. For each API, the following information displays:

Func.	Hexadecimal index number of the function passed in EAX.
Address	Selector:offset address of the start of the function.
Params	Number of dword parameters passed to the function.
Name	Either the symbolic name of the function, or the offset within NTOSKRNL if no symbols are loaded.

An example of the disassembler output follows. Note how SoftICE indicates that the INT 2Eh instruction's execution result in the NTOSKRNL function _NTSetEvent being called with 2 dword parameters.

```
ntdll!NtSetEvent
001B:77F8918C   MOV    EAX,00000095
001B:77F89191   LEA    EDX,[ESP+04]
001B:77F89195   INT    2E ; _NtSetEvent(params=02)
001B:77F89197   RET    0008
```

Example

The following example shows abbreviated output of the NTCALL command. It can be seen from this listing that the NTOSKRNL routine, _NTAccessCheck, is located at 8:80182B9Eh, that it is assigned a function identifier of 1, and that it takes 8 dword parameters.

```
00    0008:80160D42   params=06  _NtAcceptConnectPort
01    0008:80182B9E   params=08  _NtAccessCheck
02    0008:80184234   params=0B  _NtAccessCheckAndAuditAlarm
03    0008:80180C0A   params=06  _NtAdjustGroupsToken
04    0008:80180868   params=06  _NtAdjustPrivilegesToken
05    0008:8017F9A6   params=02  _NtAlertResumeThread
06    0008:8017F95E   params=01  _NtAlertThread
07    0008:8014B0C4   params=01  _NtAllocateLocallyUniqueId
08    0008:8014B39A   params=03  _NtAllocateUuids
```

O

Windows 3.1, Windows 95, Windows NT *I/O Port*

Output a value to an I/O port.

Syntax

O[***size***] ***port value***

size

Value	Description
B	Byte
W	Word
D	Dword

port Port address.

value Byte, word, or dword value as specified by size.

Use

Output to PORT commands are used to write a value to a hardware port. Output can be done to byte, word, or dword ports. If no size is specified, the default is B.

All outs are done immediately to the hardware with the exception of the interrupt mask registers (Port 21h & A1h). These do not take effect until the next time you exit from the SoftICE screen.

Example

This command performs an out to port 21, which unmasks all interrupts for interrupt controller one.

```
O 21 0
```

OBJDIR

Windows 98, Windows NT *System Information*

Displays objects in a Windows NT Object Manager's object directory.

Syntax

OBJDIR [***object-directory-name***]

Use

Use the OBJDIR command to display the named objects within the Object Manager's object directory. Using OBJDIR with no parameters displays the named objects within the root object directory. To list the objects in a subdirectory, enter the full object directory path.

Output

The following information will be displayed by the OBJDIR command:

Object	Address of the object body.
ObjHdr	Address of the object header.
Name	Name of the object.
Type	Windows NT-defined data type of the object.

Example

The following example is abbreviated output of OBJDIR listing objects in the Device object directory:

OBJDIR device

```
Directory of \Device at FD8E7F30

Object     ObjHdr     Name            Type
FD8CC750   FD8CC728   Beep            Device
FD89A030   FD89A008   NwlnkIpx        Device
FD889150   FD889128   Netbios         Device
FD8979F0   FD8979C8   Ip              Device
FD8C9ED0   FD8C9EA8   KeyboardClass0  Device
FD8C5038   FD8C5010   Video0          Device
FD8C4040   FD8C4018   Video1          Device
```

In the following example, the OBJDIR command is used with a specified object directory pathname to list the objects in the \Device\Harddisk0 subdirectory.

`OBJDIR \device\harddisk0`

```
Directory of \Device\Harddisk0 at FD8D38D0
```

Object	ObjHdr	Name	Type
FD8D3730	FD8D3708	Partition0	Device
FD8D3410	FD8D33E8	Partition1	Device
FD8D32D0	FD8D32A8	Partition2	Device

```
3 Object(s)
```

See Also

OBJTAB

OBJTAB

Windows NT *System Information*

Display entries in the WIN32 user object-handle table.

Syntax

OBJTAB [*handle* | *object-type-name* | -h]

handle Object handle.

object-type-name One of the object-type-names, predefined by SoftICE:

FREE	Free handle
HWND	Hwnd
Menu	Menu or Sub-menu object
Icon (or Crsr)	HICON or HCURSOR
DFRW	DeferWindowPos data
HOOK	Hook
TINF	Thread Info data
QUE (3.51 only)	Message queue
CPD	Call Proc Data thunk
ACCL	Accelerator table
WSTN	Workstation object
DESK(3.51 only)	Desktop object
DDE	DDE String

-h Display list of valid object-type-names.

Use

Use the OBJTAB command to display all entries in the master object-handle table created and maintained by CSRSS, or to obtain information about a specific object or objects of a certain type. The master object-handle table contains information for translating user object-handles such as an hWnd or hCursor into the actual data that represents the object.

If you use OBJTAB without parameters, SoftICE lists the full contents of the master object-handle table. If an object handle is specified, just that object is listed. If an object-type-name is entered, all objects in the master object-handle table of that type are listed.

Output

The following information is displayed by the OBJTAB command:

Object	Pointer to the object's data.
Type	Type of the object.
Id	Object's type ID.
Handle	Win32 handle value for the object.
Owner	CSRSS specific instance data for the process or thread that owns the object.
Flags	Object's flags.

Example

The following is an abbreviated example using the OBJTAB command without parameters or options:

:OBJTAB

```
Object    Type          Id      Handle    Owner     Flags
7F2D4DA0  Hwnd          01      0004005C  7F2D5F88  00
7F2D85B8  Menu          02      0001005D  00298B40  00
7F2D4E58  Hwnd          01      0003005E  7F2D5F88  00
7F2D1820  Queue         07      0002005F  00000000  00
003E50E0  Accel. Table  09      00030060  00298B40  00
```

See Also

OBJDIR

P

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

F10, F12 for P RET

Execute one program step.

Syntax

P [RET]

Use

The P command is a logical program step. In assembly mode, one instruction at the current CS:EIP is executed unless the instruction is a call, interrupt, loop, or repeated string instruction. In those cases, the entire routine or iteration is completed before control is returned to SoftICE.

If RET is specified, SoftICE will step until it finds a return or return from interrupt instruction. This function works in either 16- or 32-bit code and also works in level 0 code.

The P command uses the single step flag for most instructions. For call, interrupt, loop, or repeated string instructions, a one-time INT 3 style breakpoint execution breakpoint is used.

In source mode one source statement is executed. If the source statement involves calling another procedure, the call is not followed. The called procedure is treated like a single statement.

If the Register window is visible when SoftICE pops up, all registers that have been altered since the P command was issued will be displayed with the bold video attribute. For call instructions, this will show what registers a subroutine has not preserved.

In an unusually long procedure, there can be a noticeable delay when using the P RET command, because SoftICE is single stepping every instruction.

For Windows 95 and Windows NT

The P command, by default, is thread specific. If the current EIP is executing in thread X, SoftICE will not break until the program step occurs in thread X. This prevents the case of Windows NT process switching or thread switching during the program step causing execution to stop in a different thread or process than the one you were debugging. To change this behavior, either use the SET command with the THREADP keyword or disable thread-specific stepping in the troubleshooting SoftICE initialization settings.

Example

To execute one program step, use the command:

P

PAGE

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display page table information.

Syntax

PAGE [*address* [**L** *length*]]

address	Virtual address, segment:offset address, or selector:offset address that you want to know page table information about, including the virtual and physical address.
length	Number of pages to display.

Use

The PAGE command can be used to list the contents of the current page directory or the contents of individual page table entries.

Note: Multiple page directories are used only by Windows NT.

In the x86 architecture, a page directory contains 1024 4-byte entries, where an entry specifies the location and attributes of a page table that is used to map a range of memory related to the entry's position in the directory. (These ranges are shown on the far right in the PAGE command's output of the page directory.)

Each entry represents the location and attributes of a specific page within the memory range mapped by the page table. An x86 processor page is 4KB in size, so a page table maps 4KB/page * 1024 entries = 4MB of memory, and the page directory maps up to 4MB/page table * 1024 entries = 4GB of memory.

NT 4.0 uses the 4 MB page feature of the Pentium/Pentium Pro processors. NTOSKRNL, HAL, and all boot drivers are mapped into a 4 MB page starting at 2 GB (80000000h).

When the address parameter is specified, information about the page table entry that maps the address is shown. This includes the following:

- The linear virtual address of the start of the page mapped by the entry.
- The physical address that corresponds to the start of the page mapped by the entry.
- The page table entry attributes of the page. This information corresponds directly to processor defined attributes. Page table attributes are represented by bits that indicate whether or not the entry is valid, the page is dirty or has been accessed, whether its a supervisor or user-mode page, and its access protections. Only bit attributes that are set are shown by SoftICE.
- The page type. This information is interpreted from the Windows-defined bit field in the page table entry and the types displayed by SoftICE correspond to Windows definitions.

Use the length parameter with the address parameter to list information about a range of consecutive page table entries. It should be noted that the PAGE command will not cross page table boundaries when listing a range. This means that a second PAGE command must be used to list the pages starting where the first listing stopped, in the case that fewer entries are listed than you specified.

If no parameters are specified, the PAGE command shows the contents of the current page directory. Each line listed represents 4MB of linear address space. The first line shows the physical and linear address of the page directory. Each following line displays the information in each page directory entry. The data shown for each entry is the same as is described above for individual page table entries, however, in this output addresses represent the locations of page tables rather than pages.

Output

The following information is displayed by the PAGE command:

physical address — If a page directory is being displayed then this is the physical address of the page table that a page directory entry refers to. Each page directory entry references one page table which controls 4MB of memory.

If an address parameter is entered so that specific pages are displayed, then this is the physical address that corresponds to the start of a page.

linear address — For Windows 3.1 and Windows 95 only: If the page directory is being displayed then this is the virtual address of a page table. This is the address you would use in SoftICE to display the page table with the D command.

If specific pages are being displayed, this is the virtual address of a page. If a length was entered then this is the virtual address of the start of each page.

attribute — This is the attribute of the page directory or page table entry. The valid attributes are, as follows:

Windows 3.1, Windows 95, and Windows NT		Windows NT Only	
P	Present	**S**	Supervisor
D	Dirty	**RW**	Read/Write
A	Accessed	**4M**	4 MB page (NT 4.0 only)
U	User		
R	Read Only		
NP	Not Present		

type	For Windows 3.1 and Windows 95 only: Each page directory entry has a three-bit field that can be used by the operating system to classify page tables. Windows classifies page tables into the following six categories:

System	Private
Instance	Relock
VM	Hooked

If a page is marked Not Present, then all that is displayed is NP followed by the dword contents of the page table entry.

Example

For Windows 3.1 and Windows 95

PAGE with no parameters displays page directory information. The following is a sample PAGE command output:

PAGE

```
Page Directory Physical=002B6000 Linear=006B600
```

Physical	Linear	Attributes			Type	Linear Address Range
002B7000	006B7000	P	A	U	System	00000000-003FFFFF
00109000	00509000	P	A	U	System	00400000-007FFFFF
0010A000	0050A000	P		U	System	00800000-00BFFFFF
0010B000	0050B000	P		U	System	00C00000-00FFFFFF
0010C000	0050C000	P		U	System	01000000-013FFFFF
002B8000	006B8000	P	A	U	System	80000000-803FFFFF
00106000	00506000	P	A	U	System	80400000-807FFFFF
00107000	00507000	P		U	System	80800000-80BFFFFF
00108000	00508000	P		U	System	80C00000-80FFFFFF
002B7000	006B7000	P	A	U	System	81000000-813FFFFF

PAGE with an address specified displays the page table entry that corresponds to that address. In this example, three page table entries are shown starting with the page table entry that corresponds to address 00106018. Notice that when the length parameter is specified, the linear address is truncated to the base address of the memory page that contains address.

PAGE 00106018 l 3

Linear	Physical	Attributes		Type
00106000	00006000	P	U	VM
00107000	00007000	P	U	VM
00108000	00008000	P	U	VM

In this example PAGE can be used to find both the virtual and physical address of selector:offset address.

PAGE #585:263C

Linear	Physical	Attributes		Type
0004A89C	00218442	P	U	Instance

For Windows NT

When the Page command displays information on either PTEs or PDEs for NT 4.0, 4 MB pages are indicated by a pneumonic 4M in the Attributes field. The following sample output shows the region starting at 2 GB.

```
:PAGE
Page Directory   Physical=00030000
Physical      Attributes     Linear Address Range
00000000      P   A S RW 4M  80000000 - 803FFFFF
00400000      P   A S RW 4M  80400000 - 807FFFFF
00800000      P   A S RW 4M  80800000 - 80BFFFFF
00C00000      P   A S RW 4M  80C00000 - 80FFFFFF
01034000      P   A S RW 4M  81000000 - 813FFFFF
```

The following example is a partial listing of output from the PAGE command being executed without parameters on Windows NT 3.51 so that the page directory contents are printed.

```
:PAGE
Page Directory   Physical=00030000
Physical      Attributes   Linear Address Range
00380000      P   A U RW   00000000 - 003FFFFF
00611000      P   A U RW   77C00000 - 77FFFFFF
00610000      P   A U RW   7FC00000 - 7FFFFFFF
00032000      P   A S RW   80000000 - 803FFFFF
00034000      P   A S RW   80400000 - 807FFFFF
00035000      P   A S RW   80800000 - 80BFFFFF
00033000      P   A S RW   80C00000 - 80FFFFFF
00030000      P   A S RW   C0000000 - C03FFFFF
00040000      P   A S RW   C0400000 - C07FFFFF
00001000      P   A S RW   C0C00000 - C0FFFFFF
```

Here is an example of the PAGE command being used to display the attributes and addresses of the page that instructions are currently being executed from.

```
:PAGE eip

Linear     Physical   Attributes
80404292   00404292   P D A S RW
```

PAUSE

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Pause after each screen.

Syntax

PAUSE [**on** | **off**]

Use

The PAUSE command controls screen pause at the end of each page. If PAUSE is on, you are prompted to press any key before information scrolls off the Command window. The Enter key scrolls a single line at a time. Any other key scrolls a page at a time. The prompt displays in the status line at the bottom of the Command window.

If you do not specify a parameter, the current state of PAUSE displays.

The default is PAUSE on.

Example

The following command specifies that the subsequent Command window display will not automatically scroll off the screen. You are prompted to press a key before information scrolls off the screen.

```
PAUSE on
```

See Also

SET

PCI

Windows 95, Windows 98, Windows NT *System Information*

Dump the configuration registers for each PCI device in the system.

Syntax

PCI

Use

The PCI command dumps the registers for each PCI device in the system. Do not use this command on non-PCI systems. Many of the entries are self-explanatory, but some are not. Consult the PCI specification for more information about this output.

Example

The following example illustrates a partial sample output for the PCI command:

```
:PCI

Bus 00  Device 00   Function00
        Vendor: 8086   Intel
        Device: 1237
        Revision: 02
        Device class: 06       Bridge device
        Device subclass: 00    Host bridge
        Device sub-subclass: 00
        Interrupt line: 00Interrupt pin: 00 Min_Gnt: 00   MaxLat: 00
        Cache line size: 00    Latency timer: 40          Header type: 00BIST: 00
        I/O:0   Mem:1   BusMAST:1   Special:0             MemInv:0
        Parity:0 Wait:0   SERR:1         Back2Back:0      Snoop:0
Bus 00  Device 07   Function00
        Vendor: 8086   Intel
        Device: 7000
        Revision: 01
        Device class: 06       Bridge device
        Device subclass: 01    ISA bridge
        Device sub-subclass: 00
        Interrupt line: 00Interrupt pin: 00 Min_Gnt: 00   MaxLat: 00
        Cache line size: 00    Latency timer: 00          Header type: 80BIST: 00
        I/O:1   Mem:1   BusMAST:1   Special:1             MemInv:0
        Parity:0 Wait:0   SERR:0         Back2Back:0      Snoop:0
```

PEEK

Windows 95, Windows 98, Windows NT *Display/Change Memory*

Read from physical memory.

Syntax

PEEK*[size] address*

size	B (byte), W (word), or D (dword). Size defaults to B.
address	Physical memory address.

Use

PEEK displays the byte, word, or dword at a given physical memory location. PEEK is useful for reading memory-mapped I/O registers.

Example

The following example displays the dword at physical address FF000000:

```
PEEKD FF000000
```

See Also

PAGE, PHYS, POKE

PHYS

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display all virtual addresses that correspond to a physical address.

Syntax

PHYS ***physical-address***

physical-address	Memory address that the x86 generates after a virtual address has been translated by its paging unit. It is the address that appears on the computer's BUS, and is important when dealing with memory-mapped hardware devices such as video memory.

Use

Windows uses x86 virtual addressing support to define a relationship between virtual addresses, used by all system and user code, and physical addresses that are used by the underlying hardware. In many cases a physical address range may appear in more than one page table entry, and therefore more than one virtual address range.

SoftICE does not accept physical addresses in expressions. To view the contents of physical memory you must use the PHYS command to obtain linear addresses that can be used in expressions.

For Windows 95 and Windows NT

The PHYS command is specific to the current address context. It searches the Page Tables and Page Directory associated with the current SoftICE address context.

Example

Physical address A0000h is the start of VGA video memory. Video memory often shows up in multiple virtual address in Windows. In this example there are three different virtual addresses that correspond to physical A0000 as shown:

```
:PHYS a0000
```

```
000A0000
004A0000
80CA0000
```

POKE

Windows 95, Windows 98, Windows NT *Display/Change Memory*

Write to physical memory

Syntax

POKE*[size] address value*

size	B (byte), W (word), or D (dword). Size defaults to B.
address	Physical memory address.
value	Value to write to memory.

Use

POKE writes a byte, word, or dword value to a given physical memory location. POKE is useful for writing to memory-mapped I/O registers.

Example

The following example writes the dword value 0x12345678 to physical address FF000000:

```
POKED FF000000 12345678
```

See Also

PAGE, PEEK, PHYS

Print Screen Key

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Print contents of screen.

Syntax

PRINT SCREEN key

Use

Pressing PRINT SCREEN dumps all the information from the SoftICE screen to your printer. By default, the printer port is LPT1. Use the PRN command to change your printer port. Since SoftICE accesses the hardware directly for all of its I/O, Print Screen works only on printers connected directly to a COM or LPT port. It does not work on network printers.

If you do not want to dump to a printer, choose Save SoftICE History from the File menu in the SoftICE Loader to write the SoftICE command line window history to a file.

For Windows 95 and Windows NT

From a DOS VM, use the DLOG.EXE utility to log the SoftICE Command window information.

See Also

PRN

PRN

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Set printer output port.

Syntax

PRN [lpt*x* | com*x*]

x	Decimal number between 1 and 2 for LPT, or between 1 and 4 for COM .

Use

The PRN command allows you to send output from Print Screen to a different printer port. If no parameters are supplied, PRN displays the currently assigned printer port.

Example

This command causes Print Screen output to go to the COM1 port.

```
PRN com1
```

PROC

Windows 95, Windows 98, Windows NT *System Information*

Display summary information about any or all processes in the system.

Syntax

For Windows 95

`PROC` `[-xo] [`***`task`***`]`

For Windows NT

`PROC` `[[-xom]` ***`process-type`*** `|` ***`thread-type`***`]`

-eXtended	Display extended information for each thread.
-Objects	Display list of objects in processes handle table.
-Memory	Display information about the memory usage of a process.
task	Task name.
process-type	Process handle, process ID, or process name.
thread-type	Thread handle or thread ID.

Use

If you specify PROC with no options, summary information is presented for one or all processes in the system. The information the -Memory option provides is also included when you specify the -eXtended option for Windows NT. It is provided for convenience, because the amount of extended information displayed is quite large.

For all process (and thread) times, as well as process memory information, SoftICE uses raw values from within the OS data structures without performing calculations to convert them into standardized units.

The -Object option displays the object pointer, the object handle, and the object type for every object in the processes object handle table. Because object information is allocated from the systems pageable pool, the objects type name will not always be available. In this case, question marks (???) are displayed.

Output

For Windows 95

For each process the following summary information is provided:

Process	Task name.
pProcess	Pointer to process database (pdb).
Process ID	The Ring 3 ID of the process.
Threads	Number of threads the process owns.
Context	Address context.
DefHeap	Default heap.
DebuggeeCB	Debuggee context block.

For Windows NT

For each process the following summary information is provided:

Process	Process name.
KPEB	Address of the Kernel Process Environment Block.
PID	Process ID.
Threads	Number of threads the process owns.
Priority	Base priority of the process .
User Time	Relative amount of time the process spent executing code at user level.
Krnl Time	Relative amount of time the process spent executing code at the kernel level.
Status	Current status of the process:

- Running: The process is currently running.
- Ready: The process is in a ready to run state.
- Idle: The process is inactive.
- Swapped: The process is inactive, and its address space has been deleted.
- Transition: The process is currently between states.
- Terminating: The process is terminating.

Example

For Windows 95

This example lists all the processes in the system.

:PROC

Process	pProcess	ProcessID	Threads	Context	DefHeap	DebuggeeCB
Winword	8156ACA8	FFFC8817	00000001	C10474D4	00400000	00000000
Gdidemo	81569F04	FFFCBBBB	00000001	C1033E38	00410000	00000000
Loader32	8156630C	FFFC47B3	00000001	C10476D0	00470000	00000000
Explorer	815614C0	FFFC307F	00000002	C104577C	00440000	00000000
Mprexe	8155DFA4	FFFFFB1B	00000002	C1043340	00510000	00000000
MSGSRV32	8155D018	FFFFF4A7	00000001	C1041E28	00400000	00000000
KERNEL32	8165A31C	FFFCF87A3	00000004	C10D9EDC	00640000	00000000

This example shows extended information for GDIDEMO:

:PROC -x gdidemo

Process Information for Gdidemo at 81569F04

Type:	00000005	RefCount:	00000002	Unknown1:	00000000
pEvent:	81569FC8	TermStatus:	00000103	Unknown2:	00000000
DefaultHeap:	00410000	MemContext:	C1033E38		
Flags:	00000000				
pPSP:	0001A1A0	PSPSelector:	26E7	MTEIndex:	0019
Threads:	0001	ThrNotTerm:	0001	Unknown3:	00000000
R0threads:	0001	HeapHandle:	8155B000	K16TDB:	2816
MMFViews:	00000000	pEDB:	8156A448	pHandleTable:	8156A2C0
ParentPDB:	8156630C	MODREFlist:	8156ABB0	Threadlist:	81569FE8
DebuggeeCB:	00000000	LHFreeHead:	00000000	InitialR0ID:	00000000
&crtLoadLock:	81569F64	pConsole:	00000000	Unknown4:	C007757C
ProcDWORD0:	00003734	ProcGroup:	8156630C	ParentMODREF:	8156ABB0
TopExFilter:	00000000	PriorityBase:	00000008	Heapownlist:	00650000
HHandleBlks:	0051000C	Unknown5:	00000000	pConProvider:	00000000
wEnvSel:	19B7	wErrorMode:	0000	pEvtLdFinish	8156A2A0
UTState:	0000				

Environment Database

```
Environment:   00520020 Unknown1:                         00000000
CommandLine:   8156A500 C:\PROJECTS\GDIDEMO\Gdidemo.exe
CurrentDir:    8156A524 C:\PROJECTS\GDIDEMO
StartupInfo:   8156A53C hStdIn:                           FFFFFFFF hStdOut:    FFFFFFFF
hStdError:     FFFFFFFF Unknown2:                         00000001 InheritCon  00000000
BreakType:     00000000 BreakSem:                         00000000 BreakEvent: 00000000
BreakThreadId: 00000000 BrkHandlers:                      00000000
```

This example shows a partial listing of the objects in Kernel32:

:PROC -o kernel32

```
Handle Object   Type
1      8165A32C Process
2      8155BFFC Event
3      C103E3A4 Memory Mapped file
4      C0FFE0E0 Memory Mapped file
5      C0FFE22C Memory Mapped file
6      C0FF1058 Memory Mapped file
7      8155C01C Event
8      8155CCE4 Event
9      8155CD5C Event
A      8155CD8C Thread
B      8155D008 Event
C      C1041C04 Memory Mapped file
D      8155D870 Event
```

For Windows NT

The following is an example using the PROC command without parameters:

:PROC

Process	KPEB	PID	Threads	Pri	User Time	Krnl Time	Status
System	FD8E0020	2	14	8	00000000	00001A48	Ready
smss	FD8B9020	13	6	B	00000022	00000022	Swapped
csrss	FD8B3DC0	1F	12	D	00B416C5	00049C4E	Ready
winlogon	FD8AD020	19	2	D	00000028	00000072	Idle
services	FD8A6880	28	B	9	0000018E	0000055A	Idle
lsass	FD8A4020	2A	C	9	0000001B	00000058	Idle
spoolss	FD87ACA0	43	6	8	000000AB	000000BD	Idle
nddeagnt	FD872780	4A	1	8	00000004	0000000C	Idle
*ntvdm	FD86DDC0	50	6	9	00125B98	0003C0BE	Running
scm	FD85B300	5D	3	8	00000024	0000008A	Idle
Explorer	FD850020	60	3	D	000002DE	00000447	Ready
Idle	8016A9E0	0	1	0	00000000	00135D03	Ready

Note: The process that was active when SoftICE popped up will be highlighted. The currently active process/address context within SoftICE will be indicated by an asterisk (*).

The following is an example of using the -eXtended option for a specific process, in this case Explorer:

:PROC -x explorer

```
              Extended Process Information for Explorer(60)

KPEB:      FD850020 PID:        60 Parent:    Unknown(48)
Base Pri:         D Mem Pri:     0 Quantum:          2
Usage Cnt:        1  Win Ver:  4.00  Err. Mode:        0
Status:    Ready

Processor:       00000000  Affinity:          1
Page Directory:  011CA000  LDT Base:   00000000  LDT Limit:      0000

Kernel Time:     00000447              User Time: 000002DE
Create Time:     01BB10646E2DBE90
Exit Time:       0000000000000000

Vad Root:        FD842E28  MRU Vad:    FD842E28  Empty Vad: FD823D08
DebugPort:       00000000  ExceptPort: E118B040  SE token:  E1240450
SpinLock:        00000000  HUPEB:      00000004  UPEB:      7FFDF000

ForkInProgress:  FALSE     Thread:     00000000(0)
Process Lock:    00000001  Owner:      00000000(0)
Copy Mem Lock:   00000000  Owner:      00000000(0)

Locked Pages:    00000000  ProtoPTEs:  000000DD  Modified Pages: 000000E4
Private Pages:   0000014F  Virt Size:  013F8000  Peak Virt Size: 01894000

    ---- Working Set Information ----

Update Time:     01BB11D0D7B299C0
Data:            C0502000  Table:        C0502470
Pages:           00000879  Faults:       00000899  Peak Size: 00000374
Size:            000002AF  Minimum:      00000032  Maximum:   00000159

    ---- Non Pageable Pool Statistics ----

Quota Usage:     00000E78  Peak Usage:   00001238
Inherited Usage: 0000C093  Peak Usage:   00056555  Limit:     00080000

    ---- Pageable Pool Statistics ----

Quota Usage:     00003127  Peak Usage:   00004195
Inherited Usage: 0000C000  Peak Usage:   00004768  Limit:     000009CA

    ---- Pagefile Statistics ----

Quota Usage:     00000151  Peak Usage:   0000016E
Inherited Usage: FFFFFFFF  Peak Usage:   00000151  Limit:     00000000

    ---- Handle Table Information ----

Handle Table:    E10CE5E8  Handle Array: E1265D48  Entries:         50
```

QUERY

Windows 95, Windows 98, Windows NT *System Information*

Display the virtual address map of a process.

Syntax

QUERY [[-*x*] ***address***] | [***process-type***]

-x	Shows the mapping for a specific linear address within every context where it is valid.
address	Linear address to query.
process-type	Expression that can be interpreted as a process.

Use

The QUERY command displays a map of a single process's virtual address space or the mapping for a specific linear address. If no parameter is specified, QUERY displays the map of the current process. If a process parameter is specified, QUERY displays information about each address range in the process.

Output

For Windows 95

Under Windows 95, the QUERY command displays the following information:

Base — Pointer to the base address of the region of pages.

AllocBase — Pointer to the base address of a range of pages allocated by the VirtualAlloc function that contains the base address in the Base column.

AllocProtect — Access protection assigned when the region was initially allocated.

Size — Size, in bytes, of the region starting at the base address in which all pages have the same attributes.

State — State of the pages in the region : Commit, Free, or Reserve.

- Commit — Committed pages for which physical storage was allocated
- Free — Free pages not accessible to the calling process and available to be allocated. AllocBase, AllocProtect, Protect, and Owner are undefined.
- Reserve — Reserved pages. A range of the process's virtual address space is reserved, but physical storage is not allocated. Current Access Protection (Protect) is undefined.

Protect	Current Access protection.
Owner	Owner of the region.
Context	Address context.

For Windows NT

The QUERY command displays the following information:

Context	Address context.
Address Range	Start and end address of the linear range.
Flags	Flags from the node structure.
MMCI	Pointer to the memory management structure.
PTE	Structure that contains the ProtoPTEs for the address range.
Name	Additional information about the range. This includes the following:

- Memory mapped files will show the name of the mapped file.
- Executable modules will show the file name of the DLL or EXE.
- Stacks will be displayed as STACK(thread ID).
- Thread information blocks will be displayed as TIB(thread ID).
- Any address that the WHAT command can identify may also appear.

Example

Windows 95

The following example uses the QUERY command with no parameters to display a partial listing of the map for the current process, GDIDEMO:

: QUERY

```
Base    AllocBase AllocProt Size   State   Protect Owner
0       0         0         400000 Free    NA
400000  400000    1         7000   Commit  RO      GDIDEMO
407000  400000    1         2000   Commit  RW      GDIDEMO
409000  400000    1         2000   Commit  RO      GDIDEMO
40B000  400000    1         5000   Reserve NA      GDIDEMO
410000  410000    1         1000   Commit  RW      Heap 32
411000  410000    1         FF000  Reserve NA      Heap 32
510000  410000    1         1000   Commit  RW      Heap 32
511000  410000    1         F000   Reserve NA      Heap 32
520000  520000    4         1000   Commit  RW
521000  520000    4         F000   Reserve NA
```

The following example shows every context where base address 416000 is valid:

: QUERY -x 416000

```
Base    AllocBase AllocProt Size  State   Protect Owner    Context
416000  400000    1         F1000 Reserve NA               KERNEL32
416000  400000    1         E9000 Reserve NA      Heap 32  MSGSRV32
416000  400000    1         D000  Commit  RO      EXPLORER Explorer
416000  410000    1         F9000 Reserve NA      Heap 32  WINFILE
416000  400000    1         2000  Commit  RO      CONSOLE  Console
416000  400000    1         E9000 Reserve NA      Heap 32  WINOLDAP
416000  410000    0         EA000 Free    NA               Mprexe
416000  410000    1         FA000 Reserve NA      Heap 32  Spool32
```

The following example shows a partial listing of the virtual address map for Explorer:

: QUERY EXPLORER

```
Base    AllocBase AllocProt Size   State   Protect Owner
0       0         0         400000 Free    NA
400000  400000    1         23000  Commit  RO      EXPLORER
423000  400000    1         1000   Commit  RW      EXPLORER
424000  400000    1         11000  Commit  RO      EXPLORER
435000  400000    1         B000   Reserve NA      EXPLORER
440000  440000    1         9000   Commit  RW      Heap32
449000  440000    1         F7000  Reserve NA      Heap32
540000  440000    1         1000   Commit  RW      Heap32
541000  440000    1         F000   Reserve NA      Heap32
550000  550000    4         1000   Commit  RW
551000  550000    4         F000   Reserve NA
560000  560000    1         106000 Reserve NA
```

Windows NT

The following example uses the QUERY command to map a specific linear address for Windows NT:

:QUERY 7f2d0123

```
Context  Address Range      Flags    MMCI          PTE      Name
csrss    7F2D0000-7F5CFFFF  06000000 FD8AC128      E1191068 Heap #07
```

The following example uses the QUERY command to list the address map of the PROGMAN process for Windows NT:

:QUERY progman

```
:query progman
Address Range      Flags     MMCI      PTE       Name
00010000-00010FFF  C4000001
00020000-00020FFF  C4000001
00030000-0012FFFF  84000004                      STACK(6E)
00130000-00130FFF  C4000001
00140000-0023FFFF  8400002D                      Heap #01
00240000-0024FFFF  04000000  FF0960C8  E1249948  Heap #02
00250000-00258FFF  01800000  FF0E8088  E11B9068  unicode.nls
00260000-0026DFFF  01800000  FF0E7F68  E11BBD88  locale.nls
00270000-002B0FFF  01800000  FF0E7C68  E11B6688  sortkey.nls
002C0000-002C0FFF  01800000  FF0E7AE8  E11BBA08  sorttbls.nls
002D0000-002DFFFF  04000000  FF09F3C8  E1249E88
002E0000-0035FFFF  84000001
00360000-00360FFF  C4000001
00370000-0046FFFF  84000003                      STACK(2E)
00470000-0047FFFF  04000000  FF0DF4E8  E124AAA8
00480000-00481FFF  01800000  FF0E7DE8  E110C6E8  ctype.nls
01A00000-01A30FFF  07300005  FF097AC8  E1246448  progman.exe
77DE0000-77DEFFFF  07300003  FF0FC008  E1108928  shell32.dll
77E20000-77E4BFFF  07300007  FF0FBA08  E1110A08  advapi32.dll
77E50000-77E54FFF  07300002  FF0FADC8  E1103EE8  rpcltc1.dll
77E60000-77E9BFFF  07300003  FF0FB728  E1110C48  rpcrt4.dll
77EA0000-77ED7FFF  07300003  FF0FCE08  E11048C8  user32.dll
77EE0000-77F12FFF  07300002  FF0FD868  E110F608  gdi32.dll
77F20000-77F73FFF  07300003  FF0EE1A8  E110C768  kernel32.dll
77F80000-77FCDFFF  07300005  FF0FDB48  E1101068  ntdll.dll
7F2D0000-7F5CFFFF  03400000  FF0E2C08  E11C3068  Heap #05
7F5F0000-7F7EFFFF  03400000  FF0E8EA8  E11B77E8
7FF70000-7FFAFFFF  84000001
7FFB0000-7FFD3FFF  01600000  FF116288  E1000188  Ansi Code Page
7FFDD000-7FFDDFFF  C4000001                      TIB(2E)
7FFDE000-7FFDEFFF  C4000001                      TIB(6E)
7FFDF000-7FFDFFFF  C4000001                      SubSystem Process
```

R

Windows 3.1, Windows 95, Windows 98, Windows NT *Display/Change Memory*

Display or change the register values.

Syntax

For Windows 3.1

R [***register-name*** [[=]***value***]]

For Windows 95 and Windows NT

R [-d | *register-name* | *register-name* [=] *value*]

register-name	Any of the following: AL, AH, AX, EAX, BL, BH, BX, EBX, CL, CH, CX, ECX, DL, DH, DX, EDX, DI, EDI, SI, ESI, BP, EBP, SP, ESP, IP, EIP, FL, DS, ES, SS, CS FS, GS.
value	If register-name is any name other than FL, the value is a hexadecimal value or an expression. If register-name is FL, the value is a series of one or more of the following flag symbols, each optionally preceded by a plus or minus sign: • O (Overflow flag) • D (Direction flag) • I (Interrupt flag) • S (Sign flag) • Z (Zero flag) • A (Auxiliary carry flag) • P (Parity flag) • C (Carry flag)
-d	Displays the registers in the Command window.

Use

If no parameters are supplied, the cursor moves up to the Register window, and the registers can be edited in place. If the Register window is not currently visible, it is made visible. If register-name is supplied without a value, the cursor moves up to the Register window positioned at the beginning of the appropriate register field.

If both register-name and value are supplied, the specified register's contents are changed to the value.

To change a flag value, use FL as the register-name, followed by the symbols of the flag whose values you want to toggle. To turn a flag on, precede the flag symbol with a plus sign. To turn a flag off, precede the flag symbol with a minus sign. If neither a plus or negative sign is specified, the flag value will toggle from its current state. The flags can be listed in any order.

Example

This example sets the AH register equal to 5.

R ah=5

This example toggles the O, Z, and P flag values.

R fl=ozp

This example moves the cursor into the Register window position under the first flag field.

R fl

This example toggles the O flag value, turns on the A flag value, and turns off the C flag value.

R fl=o+a-c

RS

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

F4

Restore the program screen.

Syntax

RS

Use

The RS command allows you to restore the program screen temporarily.

This feature is useful when debugging programs that update the screen frequently. Use the RS command to redisplay your program screen. To return to the SoftICE screen, press any key.

S

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Search memory for data.

Syntax

For Windows 3.1

S [*address* L *length data-list*]

For Windows 95 and Windows NT

S [-cu][*address* L *length data-list*]

address	Starting address for search.
length	Length in bytes.
data-list	List of bytes or quoted strings separated by commas or spaces. A quoted string can be enclosed with single or double quotes.
-c	Make search case-insensitive.
-u	Search for Unicode string.

Use

Memory is searched for a series of bytes or characters that matches the data-list. The search begins at the specified address and continues for the length specified. When a match is found, the memory at that address is displayed in the Data window. and the following message is displayed in the Command window.

```
PATTERN FOUND AT location
```

If the Data window is not visible, it is made visible.

To search for subsequent occurrences of the data-list, use the S command with no parameters. The search will continue from the address where the data-list was last found, until it finds another occurrence of data-list or the length is exhausted.

The S command ignores pages that are marked not present. This makes it possible to search large areas of address space using the flat data selector (Windows 3.1/Windows 95: 30h, Windows NT: 10h).

Example

This example searches for the string 'Hello' followed by the bytes 12h and 34h starting at offset ES:DI+10 for a *length* of ECX bytes.

S es:di+10 L ecx 'Hello',12,34

This example searches the entire 4GB virtual address range for 'string'.

```
S 30:0 L ffffffff 'string'
```

SERIAL

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Redirect console to serial terminal.

Syntax

SERIAL [on [*com-port*] [*baud-rate*] | off]

com-port	Number from 1 to 4 that corresponds to COM1, COM2, COM3 or COM4. Default is COM1.
baud-rate	Baud-rate to use for serial communications. The default is to have SoftICE automatically determine the fastest possible baud-rate that can be used. The rates are 1200, 2400, 4800, 9600, 19200, 23040, 28800, 38400, 57000, 115000.

Use

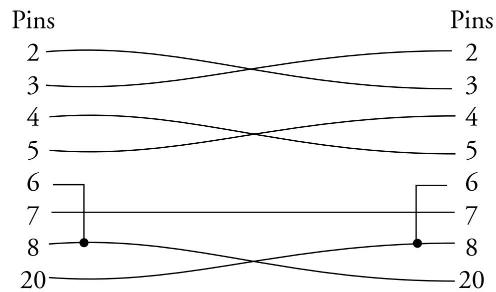
Use the SERIAL command to establish a remote debugging session through a serial port (refer to *DIAL* on page 67 for establishing remote sessions over a modem). Remote debugging requires a second IBM-compatible PC running MSDOS. The machine being debugged is known as the local machine, and the machine where SoftICE is being controlled remotely is known as the remote machine.

To use the SERIAL command, the remote and local machines must be connected with a null modem cable, with wiring as shown in the following figure, attached through serial ports. Before using the SERIAL command on the local machine, you must first run the SERIAL.EXE program on the remote machine.

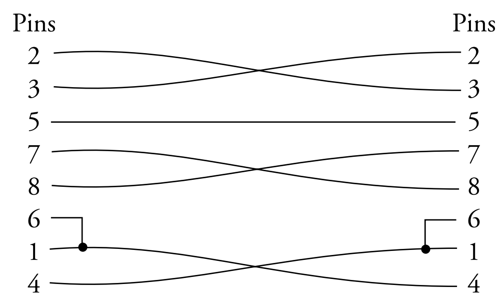
The syntax for the SERIAL.EXE program is the same as the syntax of the SERIAL command, so the following information is applicable to both.

The SERIAL command has two optional parameters. The first parameter specifies the com-port through which the connection will be made (on the machine where the command is entered). If no com-port is specified, com-port 1 (COM1) is chosen by default. The second parameter specifies a baud-rate. If a baud-rate is specified, the same baud-rate must be explicitly specified on both sides of the connection. If no baud-rate is specified, SoftICE will attempt to determine the fastest baud-rate that can be used over the connection without data loss. The process of arriving at the maximum rate can take a few seconds, during which SoftICE prints the rates it is checking. After the maximum rate is determined, SoftICE indicates the result.

When a connection is established between a remote machine and a local machine, the user of the remote machine is presented with the same SoftICE interface they would see if they were debugging on the local machine. The display on the local machine is restored to the Windows screen while the connection is maintained.



25-Pin Null-Modem Configuration



9-Pin Null-Modem Configuration

Ctrl D is always the pop-up hot key sequence on the remote machine. SoftICE can also be popped up from the local machine with the local machine's pop-up hot key sequence (which may have been set via the ALTKEY command).

If the remote machine has a monochrome display, the COLOR command can be used to make SoftICE's output more readable.

If for any reason data is lost over the connection and SoftICE output on the remote machine becomes corrupted, Shift \ (backslash) can be typed on the remote machine to force a repaint of the SoftICE screen.

Specifying SERIAL OFF will end the remote debugging session and SoftICE will resume using the local machine for I/O. SERIAL with no parameters will display the current serial state and the com-port and baud-rate being used if SERIAL is ON.

Using Ctrl-Z will exit the SERIAL.EXE program on the remote machine after a remote debugging session is complete.

If you place the SERIAL command in the SoftICE initialization string setting, SERIAL.EXE must be running on the remote machine before SoftICE is started on the local machine.

For Windows 3.1

Prior to using the SERIAL command, you must place the COM*n* keyword on a separate line in the WINICE.DAT file to reserve a specific COM port for the serial connection. The *n* is a number between 1 and 4 representing the COM port. If this statement is not present in WINICE.DAT, SoftICE cannot be popped up from the remote machine. To set Com 2 as the serial post, use:

```
Com2
```

For Windows 95

Select the desired com port in the remote debugging initialization settings within Symbol Loader.

Example

On the remote machine:

```
SERIAL.EXE on 19200
```

On the local machine:

```
SERIAL on 2 19200
```

When the first command is executed, the remote machine will be prepared to receive a connection request from the local machine on its first com-port at 19200bps. The second command establishes a connection between the two machines through the local machine's second com-port. Since the first command explicitly specified a baud rate, the SERIAL command on the local machine must explicitly specify the same baud rate of 19200bps.

Once the connection is established, the remote machine will serve as the SoftICE interface for debugging the local machine until SERIAL off is entered on the remote machine.

See Also

Chapter 7, "Debugging Remotely," in the Using SoftICE manual.

SET

Windows 95, Windows 98, Windows NT *Mode Control*

Display or change the state of an internal variable.

Syntax

SET [*keyword*] [on | off] [*value*]

Use

Use the SET command to display or change the state of internal SoftICE variables.

If you specify SET with a keyword, ON or OFF enables or disables that option. If you specify SET with a keyword and value, it assigns the value to the keyword. If SET is followed by a keyword with no additional parameters, it displays the state of the keyword.

Using SET without parameters displays the state of all keywords.

SET supports the following keywords:

ALTSCR	[on|off]
BUTTONREVERSE	[on|off]
CASESENSITIVE	[on|off]
CODE	[on|off]
EXCLUDE	[on|off]
FAULTS	[on|off]
FLASH	[on|off]
FONT	[1|2|3]
FORCEPALETTE	[on|off]
I1HERE	[on|off]
I3HERE	[on|off]
LOWERCASE	[on|off]
MOUSE	[on|off] [1|2|3]
ORIGIN	x y
PAUSE	[on|off]
SYMBOLS	[on|off]

TABS	[on|off] [1|2|3|4|5|6|7|8]
THREADP	[on|off]
VERBOSE	[on|off]
WHEELLINES	n

SET CASESENSITIVE ON makes global and local symbol names case sensitive. Enter them exactly as displayed by the SYM command.

SET MOUSE ON enables mouse support and SET MOUSE OFF disables it. To adjust the speed at which the mouse moves, use one of the following: 1 (slowest speed); 2 (intermediate speed–this is the mouse default.); 3 (fastest speed).

SET SYMBOLS ON instructs the disassembler to show the symbol names in disassembled code. SET SYMBOLS OFF instructs the disassembler to show numbers (for example, offsets and addresses). This command applies to both local and global symbol names.

Example

The following example enables SoftICE fault trapping:

```
SET faults on
```

The following example sets the mouse to the fastest speed:

```
SET mouse 3
```

See Also

ALTSCR, CODE, FAULTS, FLASH, I1HERE, I3HERE, THREADP

SHOW

Windows 3.1, Windows 95, Windows 98 *Symbol/Source*

Ctrl-F11

Display instructions from the back trace history buffer.

Syntax

SHOW [B | *start*] [l *length*]

start	Hexadecimal number specifying the index within the back trace history buffer to start disassembling from. An index of 1 corresponds to the newest instruction in the buffer.
length	Number of instructions to display.

Use

Use the SHOW command to display instructions from the back trace history buffer. If source is available for the instructions, the display is in mixed mode; otherwise, only code is displayed.

All instructions and source are displayed in the Command window. Each instruction is preceded by its index within the back trace history buffer. The instruction whose index is 1 is the newest instruction in the buffer. Once SHOW is entered, you can use the Up and Down Arrow keys to scroll through the contents of the back trace history buffer. To exit from SHOW, press the Esc key.

SHOW with no parameters or SHOW B will begin displaying from the back trace history buffer starting with the oldest instruction in the buffer. SHOW followed by a start number begins displaying instructions starting at the specified index within the back trace history buffer.

You can use the SHOW command only if the back trace history buffer contains instructions. To fill the back trace history buffer, use the BPR command with either the T or TW parameter to specifying a range breakpoint.

Example

This command starts displaying instructions in the Command window, starting at the oldest instruction in the back trace history buffer.

```
SHOW B
```

See Also

BPR

SRC

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

F3

Toggle between displaying source, mixed, and code in the Code window.

Syntax

SRC

Use

Use the SRC command to toggle among the following modes in the Code window: source mode, mixed mode, and code mode.

Hint: Use F3 to toggle modes quickly.

Example

The following example changes the current mode of the Code window:

SRC

SS

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

Search the current source file for a string.

Syntax

SS [*line-number*] ['*string*']

line-number	Decimal number.
string	Character string surrounded by quotes.

Use

The SS command searches the current source file for the specified character string. If there is a match, the line that contains the string is displayed as the top line in the Code window.

The search starts at the specified line-number. If no line-number is specified, the search starts at the top line displayed in the Code window.

If no parameters are specified, the search continues for the previously specified string.

The Code window must be visible and in source mode before using the SS command. To make the Code window visible, use the WC command. To make the Code window display source, use the SRC command.

Example

In the following example, the current source file is searched starting at line 1 for the string 'if (i==3)'. The line containing the next occurrence of the string becomes the top line displayed in the Code window.

```
SS 1 'if (i==3)'
```

STACK

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display a call stack.

Syntax

For Windows 3.1 and Windows 95

STACK [***task-name*** | ***SS:[E]BP***]

task-name	Name of the task as displayed by the TASK command.
SS:[E]BP	`SS:[E]BP` of a valid stack frame.

For Windows NT

STACK [***thread-type*** | ***stack frame***]

thread-type	Thread handle or thread ID.
stack frame	Value that is not a thread-type is interpreted as a stack frame.

Use

Use the STACK command to display the call stacks for DOS programs, Windows tasks, and 32-bit code.

If you enter STACK with no parameters, the current `SS:[E]BP` is used as a base for the stack frame displayed. You can explicitly specify a stack base with a task-name or base address, and under Windows NT, with a thread identifier.

If you are using STACK to display the stack of a Windows task that is not the current one, specify either its task-name or a valid `SS:[E]BP` stack frame. You can use the TASK command to obtain a list of running tasks. However, you should avoid using the STACK command with the current task of the TASK command's output (marked with an '*'), because the task's last known `SS:[E]BP` is no longer valid.

The STACK command walks the stack starting at the base by traversing x86 stack frames. If an invalid stack frame or address that has been paged out is encountered during the walk, the traversal will stop. The address of the call instruction at each frame is displayed along with the name of the routine it is in, if the routine is found in the current symbol table. If the routine is not in the symbol table, the export list and module name list are searched for nearby symbols. If stack variables are present, they are displayed as well.

The STACK command works in 32-bit code, however, since 32-bit symbol information support is limited to that provided in .SYM files, local variables cannot be shown. For each frame in the call stack, both the nearest symbol to the call instruction, and the actual address, are displayed. If there is no symbol available, the module name and object/section name are displayed instead.

The 32-bit call stack support is not limited to applications; it will also work for VxDs and Windows NT device driver code at ring 0. Since many VxDs are written in assembly language, there may not be a valid call stack to walk from a VxD-stack base address.

For Windows 3.1 and Windows 95, the call stack is not followed through thunks or ring transitions, but under Windows NT it is.

For Windows 3.1 and Windows 95

If you want SoftICE to pop up when a non-active task is restarted, you can use the STACK command with the task as a parameter to find the address on which to set an execution breakpoint. To do this, enter STACK followed by the task-name. The bottom line of the call stack will show an address preceded by the word 'at'. This is the address of the CALL instruction the program made to Windows that has not yet returned. You must set an execution breakpoint at the address following this call.

You can also use this technique to stop at other routines higher on the call stack. This is useful when you do not want to single step through library code until execution resumes in your program's code.

Output

Each entry of the call stack contains the following information:

- Symbol name or module name in which the return address falls
- `SS:[E]BP` value of this entry
- Call instruction's source line number if available
- Address of the first line of this routine or the name of the routine that was called to reach this routine

If stack variables are available for this entry, the following information about each is displayed:

- `SS:[E]BP` relative offset
- Stack variable name
- Data in the stack variable if it is of type char, int, or long

Example

This is the output of the STACK command after a breakpoint is set in the message handler of a Windows program.

:STACK

```
__astart at 0935:1021 [?]
WinMain at 0935:0d76 [00750]
    [BP+000C]hInstance 0935
    [BP+000A]hPrev  0000
    [BP+0006]lpszCmdLine
    [BP+0004]CmdShow
    [BP-0002]width  00DD
    [BP-0004]hWnd  00E5
USER!SENDMESSAGE+004F at 05CD:06A7
USER(01) at 0595:04A0 [?]  0595:048b
USER(06) at 05BD:1A83 [?]
=>ClockWndProc at 0935:006F [0179]
    [BP+000E]hWnd  1954
    [BP+000C]message  0024
    [BP+000A]wParam 0000
    [BP+0006]lParam  06ED:07A4
    [BP-0022]ps  0000
```

This is an example of the STACK command in 32-bit mode. Execution has been stopped within the C library DLL's memset routine:

:STACK

```
W32SCOMB!DispatchCB32+01FF at 2197:86C5003B
    UTSAMP!.text+01A4 at 2197:86C211A4
    _MyGetFreeSpace@0+0016 at 2197:86C7113B
    => MSVCRT10!memset+0005 at 2197:86C94F89
```

SYM

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

Display or set symbol.

Syntax

SYM [[***section-name***] !] ***symbol-name*** [***value***]]

section-name	Valid section-name. Also can be a partial section-name. This allows displaying symbols in a particular section. If section-name is specified, it must be followed by an exclamation point (!). For example, you could use the command `SYM .TEXT!` to display all symbols in the .TEXT section of the executable.
!	If "**!**" is the only parameter specified, the modules in this symbol table are listed.
symbol-name	Valid symbol-name. The symbol-name can end with an asterisk (*). This allows searching if only the first part of the symbol-name is known. The comma "," character can be used as a wildcard character in place of any character in the symbol-name.
value	Value that is used to set a symbol to a specific address.

Use

Use the SYM command to display and set symbol addresses. If you enter SYM without parameters, all symbols display. The address of each symbol displays next to the symbol-name.

If you specify a symbol-name without a value, the symbol-name and its address display. If the symbol-name is not found, nothing displays.

If section-name! precedes symbol-name or asterisk (*), only symbols from the specified section are shown.

The SYM command is often useful for finding a symbol when you can only remember a portion of the name. Two wildcard methods are available for locating symbols. If symbol-name ends with an asterisk (*), all symbols that match the actual characters typed prior to the asterisk display, regardless of their ending characters. If you use a comma (,) in place of a specific character in symbol-name, that character is a wild card character.

If you specify a value, the address of all symbols that match symbol-name are set to the value.

If you place an address between square brackets as a parameter to the SYM command, the closest symbol above and below the address display.

Example

All symbols that start with FOO display.

```
SYM foo*
```

All symbols that start with FOO are given the address 6000.

```
SYM foo* 6000
```

All sections for the current symbol table display.

```
SYM !
```

All symbols in section MAIN that start with FOO display.

```
SYM main!foo*
```

SYMLOC

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

Relocate the symbol base.

Syntax

For Windows 3.1

SYMLOC [***segment-address*** **| o | r |**
  (***section-number selector linear-address***)]

For Windows 95 and Windows NT

SYMLOC [***segment-address*** **| o | r | -c** ***process-type*** **|**
  (***section-number selector linear-address***)]

segment address	Only use to relocate DOS programs.
o	For 16-bit Windows table only. Changes all selector values back to their ordinal state.
r	For 16-bit Windows table only. Changes all segment ordinals to their appropriate selector value.
-c	Specify a context value for a symbol table. Use when debugging DOS extended applications.
section-number	For 32-bit tables only. PE file 1 based section-number.
selector	For 32-bit tables only. Protected mode selector.
linear-address	For 32-bit tables only. Base address of the section.

Use

The SYMLOC command handles symbol fixups in a loaded symbol table. The command contains support for DOS tables, 16-bit protected mode Windows tables (using O and R commands only), and 32-bit protected mode tables. The 32-bit support is intended for 32-bit code that must be manually fixed up such as DOS 32-bit extender applications.

In a DOS program, SYMLOC relocates the segment components of all symbols relative to the specified segment-address. This function is necessary when debugging loadable device drivers or other programs that cannot be loaded directly with the SoftICE Loader.

When relocating for a loadable device driver, use the value of the base address of the driver as found in the MAP command. When relocating for an .EXE program, the value is 10h greater than that found as the base in the MAP command. When relocating for a .COM program, use the base segment address that is found in the MAP command.

The MAP command displays at least two entries for each program. The first is typically the environment and the second is typically the program. The base address of the program is the relocation value.

For Windows 95 and Windows NT

The SYMLOC -C option allows you to associate a specific address context with the current symbol table. This option is useful for debugging an extender application under Windows NT where SoftICE would not be able to assign a context to the symbol table automatically.

Example

The following example relocates all segments in the symbol table relative to 1244. The +10 relocates a TSR that was originally an .EXE file. If it is a .COM file or a DOS loadable device driver, the +10 is not necessary.

```
:SYMLOC 1244+10
```

The following example relocates all symbols in section 1 of the table to 401000h using selector 1Bh. Each section of the 32-bit table must be relocated separately.

```
:SYMLOC 1 1b 401000
```

The following example sets the context of the current symbol table to the process whose process ID is 47. Subsequently, when symbols are used, SoftICE will automatically switch to that process.

```
:SYMLOC -c 47
```

T

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

F8

Trace one instruction.

Syntax

`T` `[=`***`start-address`***`] [`***`count`***`]`

count	Specify how many times SoftICE should single step before stopping.

Use

The T command uses the single step flag to single step one instruction.

Execution begins at the current CS:EIP, unless you specify the start-address parameter. If you specify this parameter, CS:EIP is changed to start-address prior to single stepping.

If you specify count, SoftICE single steps count times. Use the Esc key to terminate stepping with a count.

If the Register window is visible when SoftICE pops up, all registers that were altered since the T command was issued are displayed with the bold video attribute.

If the Code window is in source mode, this command single steps to the next source statement.

Example

This example single steps through eight instructions starting at memory location CS:1112.

```
T = cs:1112 8
```

TABLE

Windows 3.1, Windows 95, Windows 98, Windows NT *Symbol/Source*

Change or display the current symbol table.

Syntax

For Windows 3.1

TABLE [[**r**] ***partial-table-name***] | **autoon** | **autooff** | **$**

For Windows 95 and Windows NT

TABLE [***partial-table-name***] | **autoon** | **autooff** | **$**

partial-table-name	Symbol table name or enough of the first few characters to define a unique name.
autoon	Key word that turns auto table switching on.
autooff	Key word that turns auto table switching off.
$	Specify $ to switch to the table where the current instruction pointer is located.

Use

If you do not specify any parameters, all the currently loaded symbol tables are displayed with the current symbol table highlighted. If you specify a partial-table-name, that table becomes the current symbol table.

Use the TABLE command when you have multiple symbol tables loaded. SoftICE supports symbol tables for 16- and 32-bit Windows applications and DLLs, 32-bit Windows VxDs, Windows NT device drivers, DOS programs, DOS loadable device drivers, and TSRs.

Symbols are only accessible from one symbol table at time. You must use the TABLE command to switch to a symbol table before using symbols from that table.

If you use the AUTOON keyword, SoftICE will switch to auto table switching mode. This will cause the current table to become whichever table the instruction pointer is in when SoftICE pops up. AUTOOFF turns off this mode.

Tables are not automatically removed when your program exits. If you reload your program with the SoftICE Loader, the symbol table corresponding to the loaded program is replaced with the new one.

For Windows 3.1

If the R parameter precedes partial-table-name, the specified table is removed. Specifying an “*” after the R parameter removes all symbol tables.

For Windows 95 and Windows NT

Symbol tables can be tied to an address context or multiple address contexts. If a table is tied to a context, switching to that table using the TABLE command switches to the appropriate address context. If you use any symbol from a context sensitive table, SoftICE switches to that context. Use View Symbol Tables in SoftICE Loader to remove tables from memory. The R parameter is not supported.

Example

Since no parameters are specified in the following command, all loaded symbol tables are listed. GENERIC is highlighted, because it is the current table. The amount of available symbol table memory is displayed at the bottom.

```
:TABLE
   MYTSR.EXE
   MYAPP.EXE
   MYVXD
   GENERIC
   006412 bytes of symbol table memory available
```

In the following example, the current table is changed to MYTSR.EXE. Notice that only enough characters to identify a unique table were entered.

```
:TABLE myt
```

TABS

Windows 3.1, Windows 95, Windows 98, Windows NT *Customization*

Display or set the tab settings for source display.

Syntax

TABS [***tab-setting***]

tab-setting	Number from 1 through 8 that specifies how many columns between tab stops.

Use

Use the TABS command to display or set tab-settings for the display of source files. Tab stops can be anywhere from 1 to 8 columns. The default TABS setting is 8. TABS with no parameters display the current tab-setting. Specifying a tab-setting of 1 allows the most source to be viewed since each tab will be replaced by a single space.

Example

This example causes the tabs setting to change to every fourth column starting at the first display column.

```
TABS 4
```

TASK

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the Windows task list.

Syntax

TASK

Use

The TASK command displays information about all tasks that are currently running. The task that has focus is displayed with an asterisk after its name. This command is useful when a general protection fault occurs because it indicates which program caused the fault.

For Windows NT

The TASK command is process specific and only shows 16-bit tasks under Windows NT. In addition, it is only useful when the current context is that of an NTVDM process containing a WOW box. To view information or processes, refer to *PROC* on page 162.

Output

For each running task, the following information displays:

Task Name	Name of the task.
SS:SP	Stack address of the task when it last relinquished control.
StackTop	Top of stack offset.
StackBot	Bottom of stack offset.
StackLow	Lowest value that SP has ever had when there was a context-switch away from the task.
TaskDB	Selector for the task data base segment.
hQueue	Queue handle for the task. This is just the selector for the queue.
Events	Number of outstanding events in the queue.

For Windows 3.1 and Windows 95

The TASK command works for 16- and 32-bit tasks, however, the following fields change for 32-bit tasks:

StackBot	Highest legal address of the stack shown as a 32-bit flat offset.
StackTop	Lowest legal address of the stack shown as a 32-bit flat offset.

StackLow	Field is not used.
SS:SP	Contains the 16-bit selector offset address of the stack. If you examine the base address of the 16-bit selector, you see that this points at the same memory as does the flat 32-bit pointer used with the 32-bit data selector.

Example

The following example shows the TASK command on Windows 3.1 running Win32s and its output.

:TASK

```
TaskNm     SS:SP        StackTop   StackBot   Low    TaskDB hQueue Events
FREECELL   21BF:7D96    86CE0000   86D00000          10FF   121F   0000
PROGMAN    17A7:200A    0936       2070       14CE   064F   07D7   0000
CLOCK      1427:1916    02E4       1A4E       143E   144F   1437   0000
MSWORD     * 29AF:913E  5956       93A4       7ADE   1F67   1F47   0000
```

THREAD

Windows 95, Windows 98 *System Information*

Display thread information.

Syntax

`THREAD [TCB | ID | task-name]`

TCB	Thread Control Block.
ID	Thread ID number.
task-name	Name of a currently running 32-bit process.

Use

Use the THREAD command to obtain information about a thread.

- If you do not specify any options or parameters, the THREAD command displays information for every active thread in the system.
- If you specify a task-name as a parameter, all active threads for that process display.
- If you specify a TCB or ID, only information for that thread displays.

For Windows NT, refer to THREAD on page 201.

Output

For each thread, the following information is shown:

Ring0TCB	Address of the Ring-0 thread control block. This is the address that is passed to VxDs for thread creation and termination.
ID	VMM Thread ID.
Context	Context handle associated with the process of the thread.
Ring3TCB	Address of the KERNEL32 Ring-3 thread control block
Thread ID	Ring-3 thread ID
Process	Address of the KERNEL32 process database that owns the thread.
TaskDB	Selector of the task database that owns the thread.
PDB	Selector of the program database (protected-mode PSP).
SZ	Size of the thread which can be either 16 or 32 bit.
Owner	Process name of the owner.

If you specify TCB or ID, this information displays for the thread with that TCB or ID:

- Current register contents for the thread.
- All thread local storage offsets within the thread. This shows the offset in the thread control block of the VMM TLS entry, the contents of the TLS entry, and the owner of the TLS entry.

Example

This example displays the thread that belongs to the Winword process:

:THREAD

```
Ring0TCB  ID   Context    Ring3TCB ThreadID  Process   TaskDB  PDB  SZ  Owner
C1051808  008B C104B990   815842CC FFF0671F  8158AAA8  274E    25B7 32  *Winword
```

The following example shows abbreviated information about the thread with ID 8B.

:THREAD 8B

```
Ring0TCB ID   Context  Ring3TCB  ThreadID Process  TaskDB PDB  SZ   Owner
C1051808 008B C104B990 815842CC  FFF0671F 8158AAA8 274E   25B7 32   *Winword
CS:EIP=0137:BFF96868 SS:ESP=013F:0062FC3C DS=013F ES=013F FS=2EBF GS=0000
EAX=002A002E EBX=815805B8 ECX=815842CC EDX=815805B8 I S P
ESI=00000000 EDI=815805B8 EBP=0062FC80 ECODE=00000000
TLS Offset 007C = 00000000 VPICD
TLS Offset 0080 = 00000000 DOSMGR
TLS Offset 0084 = 00000000 SHELL
TLS Offset 0088 = C1053434 VMCPD
TLS Offset 008C = C104EA74 VWIN32
TLS Offset 0090 = 00000000 VFAT
TLS Offset 0094 = 00000000 IFSMgr
```

See Also

For Windows NT, refer to *THREAD* on page 201.

THREAD

Windows NT *System Information*

Display information about a thread.

Syntax

THREAD `[-r | -x | -u] [`***thread-type*** `|` ***process-type***`]`

-r	Display value of the thread's registers.
-x	Display extended information for each thread.
-u	Display threads with user-level components.
thread-type	Thread handle or thread id.
process-type	Process-handle, process-id or process-name.

Use

Use the THREAD command to obtain information about a thread.

For Windows 95, refer to THREAD on page 199.

- If you do not specify any options or parameters the THREAD command displays information for every active thread in the system.
- If you specify a process-type as a parameter, all the active threads for that process display.
- If you specify a thread-type, only information for that thread displays.

For the -R and -X options, the registers shown are those that are saved on thread context switches: ESI, EDI, EBX and EBP.

Output

For each thread, the following summary information is displayed:

TID	Thread ID.
Krnl TEB	Kernel Thread Environment Block.
StackBtm	Address of the bottom of the thread's stack.
StackTop	Address of the start of the thread's stack.
StackPtr	Threads current stack pointer value.
User TEB	User thread environment block.
Process(Id)	Owner process-name and process-id.

Many fields of thread environment blocks are shown when extended output is specified, with most being self-explanatory. Some are particularly useful and deserve to be highlighted:

TID	Thread ID.
KTEB	Kernel Thread Environment Block.
Base Pri, Dyn. Pri	Threads base priority and current priority.
Mode	Indicates whether the thread is executing in user or kernel mode.
Switches	Number of context switches made by the thread.
Affinity	Processor affinity mask of the thread. Bit positions that are set represent processors on which the thread is allowed to execute.
Restart	Address at which the thread will start executing when it is resumed.

The thread's stack trace is displayed last.

Example

The following example uses the THREAD command to display the threads that belong to the Explorer process:

:THREAD explorer

TID	Krnl TEB	StackBtm	StkTop	StackPtr	User TEB	Process(Id)
006A	FD857DA0	FB1CB000	FB1CD000	FB1CCED8	7FFDE000	Explorer(6B)
006F	FD854620	FB235000	FB237000	FB236B2C	7FFDD000	Explorer(6B)
007C	FD840020	FD72F000	FD731000	FD730E24	7FFDB000	Explorer(6B)

This example displays extended information on the thread with ID 5Fh:

: THREAD -x 5f

```
                     Extended Thread Info for thread 5F
    KTEB:      FD850D80  TID:       05F  Process:   Explorer(60)
    Base Pri:         D  Dyn. Pri:    E  Quantum:          2
    Mode:      User      Suspended:   0  Switches:  00024B4F
    TickCount: 00EE8DA4  Wait Irql:   0
    Status:    User Wait for WrEventPair
    Start EIP:      KERNEL32!LeaveCriticalSection+0058 (6060744C)
    Affinity:       00000001             Context Flags:       A
    KSS EBP:        FB1C3F04             Callback ESP:  00000000
    Kernel Stack:   FB1C2000 - FB1C4000  Stack Ptr:     FB1C3ED8
    User Stack:     00030000 - 00130000  Stack Ptr:     0012FE3C
    Kernel Time:    0000014A             User Time:     0000015F
    Create Time:    01BB10646E2DBE90
    SpinLock:  00000000  Service Table: 80174A40  Queue:    00000000
    SE Token:  00000000  SE Acc. Flags: 001F03FF
    UTEB:      7FFDE000  Except Frame:  0012FEB4  Last Err: 00000006
```

```
    Registers:  ESI=FD850D80  EDI=0012FEC4  EBX=77F6BA0C
       EBP=FB1C3F04
    Restart  :  EIP=80168757 a.k.a. _KiSetServerWaitClientEvent+01CF
Explorer!.text+975D at 001B:0100A75D
Explorer!.text+9945 at 001B:0100A945
Explorer!.text+A3F8 at 001B:0100B3F8
USER32!WaitMessage+004F at 001B:60A0CA4B
user32!.text+070A at 001B:60A0170A
=> ntdll!CsrClientSendMessage+0072 at 001B:77F6BA0C
```

See Also

For Windows 95, refer to *THREAD* on page 199.

TRACE

Windows 3.1, Windows 95, Windows 98 *Symbol/Source*

CTRL-F9, TRACE B, CTRL-F12

Enter or exit Trace simulation mode.

Syntax

TRACE [b | off | *start*]

start	Hexadecimal number specifying the index within the back trace history buffer to start tracing from. An index of 1 corresponds to the newest instruction in the buffer.

Use

Use the TRACE command to enter, exit, and display the current state of the trace simulation mode. TRACE with no parameters displays the current state of trace simulation mode. TRACE followed by off exits from trace simulation mode and returns to regular debugging mode. TRACE B enters trace simulation mode starting from the oldest instruction in the back trace history buffer. TRACE followed by a start number enters trace simulation mode at the specified index within the back trace history buffer.

You can use the trace simulation mode only if the back trace history buffer contains instructions. To fill the back trace history buffer, use the BPR command with either the T or TW parameter to specifying a range breakpoint.

When trace simulation mode is active, the help line on the bottom of the screen shows this, as well as the index of the current instruction within the back trace history buffer.

Use the XT, XP, and XG commands to step through the instructions in the back trace history buffer from within the trace simulation mode. When stepping through the back trace history buffer, the only register that changes is the EIP register because back trace ranges do NOT record the contents of all the registers. You can use all the SoftICE commands within trace simulation mode except for the following: X, T, G, P, HERE, and XRSET.

Example

This example enters trace simulation mode starting at the eighth instruction in the back trace history buffer.

```
TRACE 8
```

See Also

BPR, BPRW, SHOW

TSS

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display task state segment and I/O port hooks.

Syntax

For Windows 3.1

TSS

For Windows 95 and Windows NT

TSS [***TSS-selector***]

TSS-selector	Any GDT selector that represents a TSS.

Use

This command displays the contents of the task state segment after reading the task register (TR) to obtain its address.

You can display any 32-bit TSS by supplying a valid 32-bit Task Gate selector as a parameter. Use the GDT command to find TSS selectors. If you do not specify a parameter, the current TSS is shown.

Output

The following information is displayed:

TSS selector value	TSS selector number.
selector base	Linear address of the TSS.
selector limit	Size of the TSS.

The next four lines of the display show the contents of the register fields in the TSS. The following registers are displayed:

```
LDT, GS, FS, DS, SS, CS, ES, CR3
EAX, EBX, ECX, EDX, EIP
ESI, EDI, EBP, ESP, EFLAGS
Level 0, 1 and 2 stack SS:ESP
```

For Windows 3.1 and Windows 95

Next, the TSS bit mask array is printed, which shows each I/O port that has been hooked by a Windows virtual device driver (VxD). For each port, the following information is displayed:

port number	16-bit port number.
handler address	32-bit flat address of the port's I/O handler. All I/O instructions on the port will be reflected to this handler.

handler name	Symbolic name of the I/O handler for the port. If symbols are available for the VxD, the nearest symbol will be displayed; otherwise the name of the VxD followed by the handler's offset within the VxD will be displayed.

For Windows 95 and Windows NT

The I/O permission map base and size are also displayed. A size of zero indicates that all I/O is trapped. A non-zero size indicates that the I/O permission map determines if an I/O port is trapped.

Example

The following example displays the task state segment in the Command window (output of the bit mask array is abbreviated).

:TSS

```
TR=0018  BASE=C000AEBC  LIMIT=2069
LDT=0000  GS=0000  FS=0000  DS=0000  SS=0000  CS=0000  ES=0000
CR3=00000000
EAX=00000000  EBX=00000000  ECX=00000000  EDX=00000000  EIP=00000000
ESI=00000000  EDI=00000000  EBP=00000000  ESP=00000000  EFL=00000000
SS0=0030:C33EEFA8  SS1=0000:00000000  SS2=0000:00000000
I/O Map Base=0068  I/O Map Size=2000


Port  Handler   Trapped  Owner
0000  C00C3E92  Yes      VDMAD(01)+17BA
0001  C00C3F0E  Yes      VDMAD(01)+1836
0002  C00C3E92  Yes      VDMAD(01)+17BA
0003  C00C3F0E  Yes      VDMAD(01)+1836
0004  C00C3E92  Yes      VDMAD(01)+17BA
0005  C00C3F0E  Yes      VDMAD(01)+1836
0006  C00C3E92  Yes      VDMAD(01)+17BA
0007  C00C3F0E  Yes      VDMAD(01)+1836
0008  C00C3C55  Yes      VDMAD(01)+157D
0009  C00C3D98  Yes      VDMAD(01)+16C0
```

If you are interested in which VxD has hooked port 21h (interrupt mask register), you would look at the TSS bit mask output of the TSS display for the entry corresponding to the port. The following output, taken from the TSS command's output, indicates that the port is hooked by the virtual PIC device and its handler is at offset 800792B4 in the flat code segment. This corresponds to an offset of 0AF8h bytes from the beginning of VPICD's code segment.

```
0021   800792B4    VPICD+0AF8
```

TYPES

Windows 95, Windows 98, Windows NT *Symbol/Source Command*

List all types in the current context or list all type information for the type-name specified.

Syntax

TYPES [***type-name***]

type-name List all type information for the type-name specified.

Use

If you do not specify a type-name, TYPES lists all the types in the current context. If you do specify a type-name, TYPES lists all the type information for the type-name you specified. If the type-name you specified is a structure, TYPES expands the structure and lists the typedefs for its members.

Example

The following example displays a partial listing of all the types in the current context:

:TYPES

```
Size      Type Name                 Typedef
0x0004    ABORTPROC                 int stdcall (*proc) (void)
0x0004    ACCESS_MASK               unsigned long
0x0004    ACL_INFORMATION_CLASS     int
0x0018    ARRAY_INFO                struct ARRAY_INFO
0x0002    ATOM                      unsigned short
0x0048    BALLDATA                  struct _BALLDATA
0x0048    _BALLDATA                 struct _BALLDATA
0x0020    _BEZBUFFER                struct _BEZBUFFER
0x0004    BOOL                      int
0x0001    BOOLEAN                   unsigned char
0x0010    _BOUNCEDATA               struct _BOUNCEDATA
0x0004    BSTR                      unsigned short *
```

The following example displays all the type information for the type-name _bouncedata:

:TYPES _bouncedata

```
typedef struct _BOUNCEDATA {
public:
   void * hBall1 ;
   void * hBall2 ;
   void * hBall3 ;
   void * hBall4 ;
};
```

See Also

LOCALS, WL

U

Windows 3.1, Windows 95, Windows 98, Windows NT *Display/Change Memory*

Unassemble instructions.

Syntax

For Windows 3.1

U [***address***] | [***symbol-name***]

For Windows 95 and Windows NT

U [***address*** [**l** ***length***]] | [***symbol-name***]

address	Segment offset or selector offset.
symbol-name	Scrolls the Code window to the function you specify.
length	Number of instruction bytes.

Use

The U command displays either source code or unassembled code at the specified address. The code displays in the current mode (either code, mixed, or source) of the Code window,. Source displays only if it is available for the specified address. To change the mode of the Code window, use the SRC command (default key F3).

If you do not specify the address, the command unassembles at the address where you left off.

If the Code window is visible, the instructions display in the Code window, otherwise they display in the Command window. In the Command window either eight lines display, or one less than the length of the Command window.

To make the Code window visible, use the WC command (default key Alt-F3). To move the cursor to the Code window, use the EC command (default key F6).

If the instruction is at the current CS:EIP, it displays using the reverse video attribute. If the current CS:EIP instruction is a relative jump, it contains either the string JUMP or NO JUMP, indicating whether or not the jump will be taken, and if so, an arrow indicating if the jump will go up or down in the Code window. If the current CS:EIP instruction references a memory location, the contents of the memory location display in the Register window beneath the flags field. If the Register window is not visible, this value displays on the end of the code line.

If a breakpoint is set on an instruction being displayed, the code line is displayed using the bold attribute.

If any of the memory addresses within an instruction have a corresponding symbol, the symbol displays instead of the hexadecimal address. If an instruction is located at a code symbol, the symbol name displays on the line above the instruction.

To view or suppress the actual hexadecimal bytes of the instruction, use the CODE command.

For Windows 95 and Windows NT

If you specify a length, SoftICE disassembles the instructions in the Command window instead of the Code window. This is useful for reverse engineering, for example, disassembling an entire routine and then using the SoftICE Loader Save SoftICE History function to capture the output to a file.

Example

To unassemble instructions beginning at 10 hexadecimal bytes before the current address, use the command:

```
U eip - 10
```

To display source in the Code window starting at line number 121, use the command:

```
U .121
```

For Windows 95 and Windows NT

To disassemble 100 h bytes starting at MyProc to the Command window, use the command:

```
U myproc L100
```

VCALL

Windows 3.1, Windows 95, Windows 98 *System Information*

Display the names and addresses of VxD callable routines.

Syntax

VCALL [***partial-name***]

partial-name	VxD callable routine name or the first few characters of the name. If more than one routine's name matches the partial-name, all routines that start with the specified characters are listed.

Use

The VCALL command displays the names and addresses of Windows VxD API routines. These are Windows services provided by VxDs for other VxDs. All the routines SoftICE lists are located in Windows system VxDs that are included as part of the base-line Windows kernel.

The addresses displayed are not valid until the VMM VxD is initialized. If an X is not present in the SoftICE initialization string, SoftICE pops up while Windows is booting and VMM is not initialized.

The names of all VxD APIs are static. Only the function names provided in the Windows DDK Include Files are available. These API names are not built into the final VxD executable file. SoftICE provides API names for the following VxDs:

CONFIGMG	IOS	VCD	VMCPD	VSD
DOSMGR	NDIS	VCOMM	VMD	VTD
DOSNET	PAGEFILE	VCOND	VMM	VWIN32
EBIOS	PAGESWAP	VDD	VMPOLL	VXDLDR
ENABLE	SHELL	VDMAD	VNETBIOS	
IFSMGR	V86MMGR	VFBACKUP	VPICD	
INT13	VCACHE	VKD	VREDIR	

Example

The following example lists all Windows system VxD calls that start with Call. Sample output follows the command.

VCALL call

```
80006E04        Call_When_VM_Returns
80009FD4        Call_Global_Event
80009FF4        Call_VM_Event
8000A018        Call_Priority_VM_Event
8000969C        Call_When_VM_Ints_Enabled
800082C0        Call_When_Not_Critical
8000889F        Call_When_Task_Switched
8000898C        Call_When_Idle
```

VER

Windows 3.1, Windows 95, Windows 98, Windows NT *Miscellaneous*

Display the SoftICE version number.

Syntax

VER

Hint: To view your registration information and product serial number, start SoftIce Loader and choose About SoftICE Loader from the Help menu.

Example

The following example displays the SoftICE version number and operating system version:

VER

VM

Windows 3.1, Windows 95, Windows 98 *System Information*

Display information on virtual machines.

Syntax

VM *[-S] [VM-ID]*

-S	Switches to the VM identified by the VM-ID.
VM-ID	Index number of the virtual machine. Index numbers start at 1, where index number 1 is always assigned to the Windows System VM (the VM in which Windows applications run).

Use

If no parameters are specified, the VM command displays information about all virtual machines (VM) in the system. If a VM-ID is specified, the register values of the VM are displayed. These registers are those found in the client register area of the virtual machine control block so they represent the values last saved into the control block when there was a context switch away from the VM. If SoftICE is popped up while a VM is executing, the registers displayed in the SoftICE Register window, not the ones shown in the VM command output, are the current registers for the VM. However, if you are in the first few instructions of an interrupt routine where a virtual machine's registers are being saved to the control block, the CS:IP register may be the only valid register (the others have not been saved yet).

The command displays two sets of segment registers plus the EIP and SP registers. The segment registers are used for the protected mode and the real mode contexts of the VM. If a VM was executing in protected mode last, the protected mode registers are listed first. If V86 mode was the last execution mode, the V86 segment registers are listed first. The general purpose registers (displayed below the segment registers) pertain to the segment registers listed first.

A VM is a unit of scheduling for the Windows kernel. A VM can have one protected mode thread under Windows 3.1, and multiple protected mode threads under Windows 95. In both cases the VM has one V86 mode thread of execution. Windows, Windows applications, and DLLs all run in protected mode threads of VM 1 (the System VM).

VMs other than the System VM normally have a V86 thread of execution only. However, DPMI applications (also known as DOS extended applications) launched from these VMs can also execute in a protected mode thread.

The VM command is very useful for debugging VxDs, DPMI programs, and DOS programs running under Windows. For example, if the system hangs while running a DOS program, you can often find the address of the last instruction it executed with the VM command (the CS:EIP shown for the VM's V86 thread).

Another more esoteric, but highly valuable use for the VM command is found when Windows faults all the way back to DOS. There are times when Windows cannot handle a fault and exits Windows and you end up back at the DOS prompt.

If this happens, duplicate the problem with I1HERE ON in SoftICE (Windows executes an INT 1 prior to returning to DOS). When the fault happens, SoftICE pops up. Use the VM command to find out the last address of execution and use the CR command to find the fault address (CR2 contains the fault address). The ESI register usually points to an error message at this point.

Output

For each virtual machine, the following information displays:

VM Handle VM handle is actually a flat offset of the data structure that holds information about the VM.

Status This is a bit mask that shows current state information for the VxD. The values are as follows:

`0001H`	`Exclusive mode`
`0002H`	`Runs in background`
`0004H`	`In process of creating`
`0008H`	`Suspended`
`0010H`	`Partially destroyed`
`0020H`	`Executing protected mode code`
`0040H`	`Executing protected mode app`
`0080H`	`Executing 32-bit protected app`
`0100H`	`Executing call from VxD`
`0200H`	`High priority background`
`0400H`	`Blocked on semaphore`
`0800H`	`Woke up after blocked`
`1000H`	`Part of V86 App is pageable`
`2000H`	`Rest of V86 is locked`
`4000H`	`Scheduled by time-slices`
`8000H`	`Idle, has released time slice`

High Address Alternate address space for VM. This is where a VxD typically accesses VM memory (instead of 0).
Note: It is likely for parts of the VM to be paged out at any one time

	that you pop up SoftICE.
VM-ID	Index number of this VxD, starting at 1.
Client Registers	Address of the saved registers of this VM. This address actually points into the level 0 stack for this VM.

Example

`VM`

Sample output follows:

```
VM Handle   Status      High Addr   VM-ID       Client Regs
806A1000    00004000    81800000    3           806A8F94
8061A000    00000008    81400000    2           80515F94
80461000    00007060    81000000    1           80013390
```

VXD

Windows 3.1 *System Information*

Display the Windows VxD map.

Syntax

VXD [*VxD-name* | *partial-VxD-name*]

VxD-name	Name of a virtual device driver.
partial-VxD-name	First few characters of the name.

Use

This command displays a map of all Windows virtual device drivers in the Command window. If no parameters are specified, all VxDs are displayed. If a VxD-name is specified, only information about the VxD with that name displays.

For Windows 95, refer to VXD on page 218.

Information that is shown about a VxD includes the VxD's control procedure address, its Protected Mode and V86 API addresses, and the addresses of all VxD services it implements. If the current CS:EIP belongs to one of the VxD's in the map, the line with the address range that contains the CS:EIP will be highlighted.

If a partial name is specified, SoftICE displays information on all VxDs whose name begins with the partial name.

Output

If no parameters are specified, each entry in the VxD map contains the following information:

VxD name	Name specified in the .DEF file when the VxD was built.
address	Flat 32-bit address of one VxD section. VxDs are comprised of multiple sections where each section contains both code and data. (i.e. LockCode, LockData would be one section.)
size	Length of the VxD section. This includes both the code and the data of the VxD group.
code selector	Flat code selector.
data selector	Flat data selector.
type	Section number from the .386 file.
id	VxD ID number. The VxD ID numbers are used to obtain the Protected Mode and V86 API addresses that applications call.
DDB	Address of the VxDs Device Descriptor Block (DDB). This is a control block that contains information about the VxD such as the address of the Control Procedure and addresses of APIs.

If a VxD name is specified, the following information is displayed in addition to the previous information:

Control Procedure	Routine to which all VxD messages are dispatched.
Protected Mode API	Address of the routine where all services called by protected mode applications are processed.
V86 API Address	Address of the routine where all services called by V86 applications are processed.
VxD Services	List of all VxD services that are callable from other VxDs. For the Windows system VxDs, both the name and the address of the routines are displayed.

Example

This example displays the VxD map in the Command window. The first few lines of the display would look something like the following. The VxD names in the previous table can be used as symbol names. The address of seg 1 will be used when a VxD name is used in an expression.

:VXD

VxDName	Address	Length	Code	Data	Type	ID	DDB
VMM	80001000	000193D0	0028	0030	LGRP	01	
VMM	80200000	00002F1C	0028	0030	IGRP		
LoadHi	8001A3d0	000007E8	0028	0030	LGRP	02	
LoadHi	80202F1C	00000788	0028	0030	IGRP		
WINICE	8001ABB8	00027875	0028	0030	LGRP		
CV1	80042430	0000036B	0028	0030	LGRP		
VDDVGA	8004279C	00007AD8	0028	0030	LGRP		
VDDVGA	802036A8	000005EC	0028	0030	IGRP		

See Also

For Windows 95, refer to *VXD* on page 218.

VXD

Windows 95, Windows 98 *System Information*

Display the Windows VxD map.

Syntax

VXD [***VxD-name***]

VxD-name	Name or partial name of one or more virtual device drivers.

Use

Use this command to obtain information about one or more VxDs. If you do not specify any parameters, it displays a map of all the Windows virtual device drivers that are currently loaded in the system. Dynamically loaded VxDs are listed after statically loaded VxDs. If a VxD-name is specified, only that VxD, or VxDs with the same string at the start of their name are displayed. For example, VM will match VMM and VMOUSE. If the current CS:EIP belongs to one of the VxDs in the map, the line with the address range that contains the CS:EIP is highlighted.

For Windows 3.1, refer to VXD on page 216.

If no parameters are specified, each entry in the VxD map contains this information:

VxDName	VxD Name.
Address	Base address of the segment.
Length	Length of the segment.
Seg	Section number from the executable.
ID	VxD ID.
DDB	Address of the VxD descriptor block.
Control	Address of the control dispatch handler.
PM	Y, if the VxD has a protected mode API. N otherwise.
V86	Y, if the VxD has a V86 API. N otherwise.
VXD	Number of VxD services implemented.
Win32	Number of Win32 services implemented.

If a unique VxD name is specified, the following additional information appears:

Init Order	Order in which VxDs receive control messages. A zero value indicates highest priority.
Reference Data	The dword value that was passed from the real mode initialization procedure (if any) of the VxD.

Version	VxD version number.
PM API	PM API FLAT procedure address and PM API Ring-3 address used by applications. Refer to the following comments on PM and V86 APIs.
V86 API	V86 API FLAT procedure address and V86 API Ring-3 address used by applications. Refer to the next comments on PM and V86 APIs.

The PM API and V86 API parameters are register based and it is up to the individual VxD to define subfunctions and parameter passing (on entry EBX-VM Handle, EBP-client registers). If the Ring-3 address shown is 0:0, it means that no application code has yet requested the API address through INT 2F function 1684h.

When the VxD being listed has a Win32 service table, the following information is presented for each service:

Service Number	Win32 Service Number.
Service Address	Address of the service API handler.
Params	Number of dword parameters the service requires.

When the VxD being listed has a VxD service table, the following is shown for each service:

Service Number	VxD service number.
Service Address	Flat address of service.
Service Name	Symbol name if known (from VCALL list).

Example

This example displays the VxD map in the Command window. The first few lines of the display look similar to the following. The VxD names in the previous table can be used as symbol names. The address of Seg 1 is used when a VxD name is used in an expression.

:VXD

VxD Name	Address	Length	Seg	ID	DDB	Control	PM	V86	VxD	Win32
VMM	C0001000	00FDC0	0001	0001	C000E990	C00024F8	Y	Y	402	41
VMM	C0200000	000897	0002							
VMM	C03E0000	000723	0003							
VMM	C0320000	000095	0004							
VMM	C0360000	00ED50	0005							
VMM	C0260000	007938	0006							

See Also

For Windows 3.1, refer to *VXD* on page 216.

WATCH

Windows 3.1, Windows 95, Windows 98, Windows NT *Watch*

Add a watch expression.

Syntax

WATCH *expression*

Use

Use the WATCH command to display the results of expressions. SoftICE determines the size of the result based on the expression's type information. If SoftICE cannot determine the size, dword is assumed. The expressions being watched are displayed in the Watch window. There can be up to eight watch expressions at a time. Every time the SoftICE screen is popped up, the Watch window displays the expression's current values.

Each line in the Watch window contains the following information:

- Expression being evaluated.
- Expression type.
- Current value of the expression displayed in the appropriate format.

A plus sign (+) preceding the type indicates that you can expand it to view more information. To expand the type, either double-click the type or press Alt-W to enter the Watch window, use the UpArrow and DownArrow keys to move the highlight bar to the type you want to expand, and press Enter.

If the expression being watched goes out of scope, SoftICE displays the following message: "Error evaluating expression".

To delete a watch, use either the mouse or keyboard to select the watch and press Delete.

Example

This example creates an entry in the Watch window for the variable hInstance.

```
WATCH hInstance
```

This example indicates that the type for hInstance is void pointer (void *) and its current value is 0x00400000.

```
hPrevInstance  void * = 0x00400000
```

The following example displays the dword to which the DS:ESI registers point.

```
WATCH ds:esi
   ds:esi  void * =0x8158D72E
```

To watch what ds:esi points to, use the pointer operator (*):

```
WATCH * ds:esi
```

The following example sets a watch on a pointer to a character string lpszCmdLine. The results show the value of the pointer (0x8158D72E) and the ASCII string (currently null).

```
WATCH lpszCmdLine +char * =0x8158D72E <"">
```

Double-clicking on this line expands it to show the actual string contents.

```
lpszCmdLine -char * =0x8158D72E
    char = 0x0
```

See Also

Alt-W, WW

WC

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Alt-F3

Toggles the Code window open or closed; and sets the size of the Code window.

Syntax

WC [***window-size***]

window-size Decimal number.

Use

If you do not specify window-size, WC toggles the window open or closed. If the Code window is closed, WC opens it; and if it is open, WC closes it.

If you specify the window-size, the Code window is resized. If it is closed, WC opens it to the specified size.

When the Code window is closed, the extra screen lines are added to the Command window. When the Code window is opened, the lines are taken from the other windows in the following order: Command and Data.

If you wish to move the cursor to the Code window, use the EC command (default key F6).

Example

If the Code window is closed, the following example displays the window and sets it to twelve lines. If the Code window is open, the example sets it to twelve lines.

```
WC 12
```

WD

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Alt-F2

Toggles the Data window open or closed; and sets the size of the Data window.

Syntax

WD [***window-size***]

window-size	Decimal number.

Use

If you do not specify the window-size, WD toggles the Data window open or closed. If the Data window is closed, WD opens it; and if it is open, WD closes it.

If you specify the window-size, the Data window is resized. If it is closed, WD opens it to the specified size.

When the Data window is closed, the extra screen lines are added to the Command window. When the Data window is opened, the lines are taken from the other windows in the following order: Command and Code.

If you wish to move the cursor to the Data window to edit data, use the E command.

Example

If the Data window is closed, the following example displays the window and sets it to one line. If the Data window is open, the example sets it to one line.

```
WD 1
```

WF

Windows 95, Windows 98, Windows NT *Window Control*

CTRL-F3

Display the floating point stack in either floating point or MMX format.

Syntax

`WF [-d] [b | w | d | f | *]`

-d	Display the floating point stack in the Command window. In addition to the registers, both the FPU status word and the FPU control word display in ASCII format.
b	Display the floating point stack in byte hexadecimal format.
w	Display the floating point stack in word hexadecimal format.
d	Display the floating point stack in dword hexadecimal format.
f	Display the floating point stack in 10-byte real format.
*	Display the "next" format. The "*" keyword is present to allow cycling through all the display formats by pressing a function key.

Use

WF with no parameters toggles the display of the floating point Register window. The window occupies four lines and is displayed immediately below the Register window. In 10 byte real format, the registers are labeled ST0-ST7. In all other formats the registers are labeled MM0-MM7.

If the floating point stack contains an unmasked exception, SoftICE will NOT display the stack contents. When reading the FPS, SoftICE obeys the tag bits and displays 'empty' if the tag bits specify that state.

When displaying in the Command window, SoftICE displays both the status word and the control word in ASCII format.

Example

`WF -d f`

```
FPU Status Word: top=2
FPU Control Word: PM UM OM ZM DM IM pc=3 rc=0
ST0  1.619534411708533451e-289
ST1  9.930182991407099205e-293
ST2  6.779357630001165015e-296
ST3  4.274541060856685014e-299
```

```
ST4  2.782904336495237639e-302
ST5  1.818657819582844735e-305
ST6  empty
ST7  empty
```

Note: ASCII flags are documented in the *INTEL Pentium Processor User's Manual*, "Architecture and Programming," Volume 3.

When displaying in any of the hexadecimal formats, SoftICE always display left to right from most significant to least significant. For example, in word format, the following order would be used:

Word format: `bits(63-48) bits(47-32) bits(31-16) bits(15-0)`

WHAT

Windows 95, Windows 98, Windows NT *System Information*

Determine if a name or expression is a “known” type.

Syntax

WHAT [*name* | *expression*]

name	Any symbolic name that cannot evaluate as an expression.
expression	Any expression that can be interpreted as an expression.

Use

The WHAT command analyzes the parameter specified and compares it to known names/values, enumerating each possible match, until no more matches can be found. Where appropriate, type identification of a match is expanded to indicate relevant information such as a related process or thread.

The name-type parameter is typically a collection of alphanumeric characters that represent the name of an object. For example, Explorer would be interpreted as a name, and might be identified as either a module, a process, or both.

The expression-type parameter is something that would not generally be considered a name-type. That is, it is a number, a complex expression (an expression which contains operators, such as Explorer + 0), or a register name. Although a register looks like a name, registers are special cased as expressions since this usage is much more common. For example, for `WHAT eax`, the parameter eax is interpreted as an expression-type. Symbol names are treated as names, and will be correctly identified by the WHAT command as symbols.

Because the rules for determining name- and expression-types can be ambiguous at times, you can force a parameter to be evaluated as a name-type by placing it in quotes. For example, for `WHAT “eax”`, the quotes force eax to be interpreted as a name-type. To force a parameter that might be interpreted as a name-type to an expression-type, use the unary “+” operator. For example, for `WHAT +Explorer`, the presence of the unary “+” operator forces Explorer to be interpreted as a symbol, instead of a name.

Example

The following is an example of using the WHAT command on the name Explorer and the resulting output. From the output, you can see that the name Explorer was identified twice: once as a kernel process and once as a module.

`WHAT explorer`

```
The name (explorer) was identified and has the value FD854A80
   The value (FD854A80) is a Kernel Process (KPEB) for Explorer(58)

The name (explorer) was identified and has the value 1000000
   The value (1000000) is a Module Image Base for 'Explorer'
```

WL

Windows 95, Windows 98, Windows NT *Window Control Command*

Toggles the Locals window open or closed; and sets the size of the Locals window.

Syntax

WL [***window-size***]

window-size Decimal number.

Use

If you do not specify the window-size, WL toggles the Locals window open or closed. If the Local window is closed, WL opens it; and if it is open, WL closes it.

If you specify the window-size, the Locals window is resized. If it is closed, WL opens it to the specified size.

When the Locals window is closed, the extra screen lines are added to the Command window. When the Locals window is opened, the lines are taken from the other windows in the following order: Command and Code.

Hint: From within the Locals window, you can expand structures, arrays, and character strings to display their contents. Simply double-click the item you want to expand. Note that expandable items are delineated with a plus (+) mark.

Example

If the Locals window is closed, the following example displays the window and sets it to four lines. If the Locals window is open, the example sets it to four lines.

```
WL 4
```

See Also

LOCALS, TYPES

WMSG

Windows 3.1, Windows 95, Windows 98, Windows NT *System Information*

Display the names and message numbers of Windows messages.

Syntax

For Windows 3.1

WMSG [***partial-name***]

For Windows 95 and Windows NT

WMSG [***partial-name***| ***msg-number***]

partial-name	Windows message name or the first few characters of a Windows message name. If multiple Windows messages match the partial-name then all messages that start with the specified characters display.
msg-number	Hexadecimal message number of the message. Only the message that matches the msg-number displays.

Use

This command displays the names and message numbers of Windows messages. It is useful when logging or setting breakpoints on Windows messages with the BMSG command.

Example

This command displays the names and message numbers of all Windows messages that start with "WM_GET".

WMSG wm_get*

A sample output for this command follows:

```
000D   WM_GETTEXT
000E   WM_GETTEXTLENGTH
0024   WM_GETMINMAXINFO
0031   WM_GETFONT
0087   WM_GETDLGCODE
```

WMSG 111

```
0111   WM_Command
```

WR

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

F2

Toggle the Register window.

Syntax

`WR`

Use

The WR command makes the Register window visible if it is not currently visible. If the Register window is currently visible, WR closes the Register window.

The Register window displays the 80386 register set and the processor flags.

When the Register window is closed, the extra screen lines are added to the Command window.

When the Register window is made visible, the lines are taken from the other windows in the following order: Command, Code and Data.

For Windows 95 and Windows NT

The WR command also toggles the visibility of the floating point Register window if one is open.

WW

Windows 3.1, Windows 95, Windows 98, Windows NT *Window Control*

Alt-F4

Toggles the Watch window open or closed; and sets the size of the Watch window.

Syntax

WW [***window-size***]

window-size Decimal number.

Use

If you do not specify the window-size, WW toggles the Watch window open or closed. If the Watch window is closed, WW opens it; and if it is open, WW closes it.

If you specify the window-size, the Watch window is resized. If it is closed, WW opens it to the specified size.

When the Watch window is closed, the extra screen lines are added to the Command window. When the Watch window is opened, the lines are taken from the other windows in the following order: Command, Code, and Data.

Example

If the Watch window is closed, the following example displays the window and sets it to four lines. If the Watch window is open, the example sets it to four lines.

```
WW 4
```

See Also

Alt-W, WATCH

X

Windows 3.1, Windows 95, Windows 98, Windows NT *Flow Control*

F5

Exit from the SoftICE screen.

Syntax

X

Use

The X command exits SoftICE and restores control to the program that was interrupted to bring up SoftICE. The SoftICE screen disappears. If you had set any breakpoints, they become active.

Note: While in SoftICE, pressing the hot key sequence (default key Ctrl-D) or entering the G command without any parameters is equivalent to entering the X command.

XFRAME

Windows 95, Windows 98, Windows NT *System Information*

Display exception handler frames that are currently installed.

Syntax

XFRAME [***except-frame**** | ***thread-type***]

*except-frame**	Stack pointer value for an exception frame.
thread-type	Value that SoftICE recognizes as a thread.

Use

Exception frames are created by Microsoft's Structured Exception Handling API (SEH). Handlers are instantiated on the stack, so they are context specific.

When an exception handler is installed, information about it is recorded in the current stack frame. This information is referred to as an ExceptionRegistration. The XFRAME command understands this information, and walks backwards through stack frames until it reaches the top-most exception handler. From there it begins displaying each registration record up to the currently active scope. From each registration, it determines if the handler is active or inactive; its associated "global exception handler;" and if the handler is active, the SEH type: try/except or try/finally: In the case of active exception handlers, it also displays the exception filter or finally handler address.

Note: The global exception handler is actually an exception dispatcher that uses information within an exception scope table to determine which, if any, exception handler handles the exception. It also handles other tasks such as global and local unwinds.

You can use the global exception handler, and try/except/finally addresses to trap SEH exceptions by setting breakpoints on appropriate handler addresses.

The XFRAME command is context-sensitive, so if you do not specify one of the optional parameters, SoftICE reverts to the context that was active at pop-up time and displays the exception frames for the current thread. When specifying an exception frame pointer as an optional parameter, make sure you are in a context where that exception frame is valid. For thread-type parameters, SoftICE automatically switches to the correct context for the thread.

Below the information for the ExceptionRegistration record, each active handler for that exception frame is listed. For each active handler, its type (try/except or try/finally), the address of its exception filter (for try/except only), and the address of the exception handler display. Because exception handlers can be nested, more than one entry may be listed for each ExceptionRegistration record.

The XFRAME command uses bare addresses in its output. You can use either the STACK or WHAT commands to get an idea of which APIs installed which exception handlers.

Do not confuse the xScope value with the nesting level of exception handlers. Although these values may appear to have some correlation, the value of xScope is simply an index into a scope table (xTable). The scope table entry contains a link to its parent scope (if any).

In the event that a stack frame is not present, the XFRAME will not be able to complete the stack walk.

Output

For each exception frame that is installed, the following information displays:

xFrame	Address of the ExceptionRegistration. This value is stack based.
xHandler	Address of the global exception handler which dispatches the exception to the appropriate try/except/finally filter/handler.
xTable	Address of the scope table used by the global exception handler to dispatch exceptions.
xScope	Index into the xTable for the currently active exception handler. If this value is -1, the exception handler is installed, but is inactive and will not trap an exception.

Example

The following example illustrates the use of the XFRAME command for the currently active thread:

:XFRAME

```
xFrame     xHandler      xTable        xScope
------     --------      ------        ------
0x45FFFDC  0x60639638    0x606018B8    00
            try/except (0000) filter=0x60606F72, handler=0x60606F85
0x45FFFA8  0x5FE16890    0x5FE11210    00
            try/except (0000) filter=0x5FE125EB, handler=0x5FE125F8
0x45FFB74  0x77F8B1BC    0x77F61370    00
            try/except (0000) filter=0x77F7DD21, handler=0x77F7DD31
```

XG

Windows 3.1, Windows 95, Windows 98 *Symbol/Source*

Go to an address in trace simulation mode.

Syntax

`XG [r]` ***`address`***

Use

XG does a Go to a specific code address within the back trace history buffer. This command can only be used in trace simulation mode. The R parameter makes XG go backwards within the back trace history buffer. If the specified address is not found within the back trace history buffer, an error displays.

Example

This example makes the instruction at address CS:2FF000h the current instruction in the back trace history buffer.

```
XG 2ff000
```

XP

Windows 3.1, Windows 95, Windows 98 *Symbol/Source*

Ctrl-F10

Program step in trace simulation mode.

Syntax

XP

Use

The XP command does a program step of the current instruction in the back trace history buffer. It can only be used in trace simulation mode. Use this command to skip over calls to procedures and rep string instructions.

Example

This example does a program step over the current instruction in the back trace history buffer.

XP

XRSET

Windows 3.1, Windows 95, Windows 98 *Symbol/Source Command*

Reset the back trace history buffer.

Syntax

XRSET

Use

XRSET clears all information from the back trace history buffer. It can only be used when NOT in trace simulation mode.

Example

This example clears the back trace history buffer.

XRSET

XT

Windows 3.1, Windows 95, Windows 98 *Symbol/Source Command*

Ctrl-F8, XT R Alt-F8

Single step in trace simulation mode.

Syntax

XT [R]

Use

Use the XT command to single step the current instruction in the back trace history buffer. The XT command is valid only within the in trace simulation mode. This command steps to the next instruction contained in the back trace history buffer. The command XT R single steps backwards within the back trace history buffer.

Example

This example single steps one instruction forward in the back trace history buffer.

XT

ZAP

Windows 3.1, Windows 95, Windows 98, Windows NT *Mode Control Command*

Replace an embedded interrupt 1 or 3 with a NOP.

Syntax

ZAP

Use

The ZAP command replaces an embedded interrupt 1 or 3 with the appropriate number of NOP instructions. This is useful when the INT 1 or INT 3 is placed in code that is repeatedly executed and you no longer want SoftICE to pop up. This command works only if the INT 1 or INT 3 instruction is the one before the current CS:EIP.

Example

The embedded interrupt 1 or interrupt 3 will be replaced with NOP instructions in the following example:

ZAP