



COMPUWARE®

Technical support is available from our Technical Support Hotline or via our FrontLine Support Web site.

Technical Support Hotline:
1-800-538-7822

FrontLine Support Web Site:
<http://frontline.compuware.com>

This document and the product referenced in it are subject to the following legends:

Access is limited to authorized users. Use of this product is subject to the terms and conditions of the user's License Agreement with Compuware Corporation.

© 2004 Compuware Corporation. All rights reserved. Unpublished - rights reserved under the Copyright Laws of the United States.

U.S. GOVERNMENT RIGHTS

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in Compuware Corporation license agreement and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii)(OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable. Compuware Corporation.

This product contains confidential information and trade secrets of Compuware Corporation. Use, disclosure, or reproduction is prohibited without the prior express written permission of Compuware Corporation.

DriverStudio, SoftICE Driver Suite, DriverNetworks, DriverWorks, TrueCoverage, and DriverWorkbench are trademarks of Compuware Corporation. BoundsChecker, SoftICE, and TrueTime are registered trademarks of Compuware Corporation.

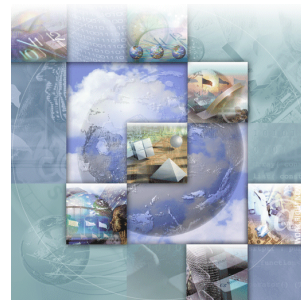
Acrobat® Reader copyright © 1987-2004 Adobe Systems Incorporated. All rights reserved. Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.

All other company or product names are trademarks of their respective owners.

US Patent Nos.: Not Applicable.

Doc. 11580b
October 13, 2004

Table of Contents



Preface

Accessibility vi

About Inline Editing . . . vi

SoftICE Commands

. 1

? 2

A 3

ADDR 5

ALTKEY 8

ALTSCR 9

ANSWER 11

APC 13

ATTACH 15

BC 18

BD 19

BE 20

BH 21

BL 22

BMSG 24

BPE 26

BPINT 28

BPIO 30

BPLOAD 32

BPM 34

BPT 37

BPX 39

BSTAT 42

C 44

CLASS 45

CLS 47

CODE 48

COLOR 49

CPU 51

CSIP 54

D 56

DATA 59

DETACH 60

DEVICE 61

DEVNODE 64

DEX 66

DIAL 68

DPC 70

DRIVER 71

E 74

EC 76

ERESOURCE 78

EVENT 80

EVMEM 84

EVRES 86

EXP 89

F 91

FAULTS 92

FIBER 93

FILE 94

FILTER 96

FKEY 100

FLASH 103

FMUTEX 104

FOBJ 105

FORMAT 107

G 108

GDT 109

GENINT 112

H 114

HBOOT 115

HEAP 116

HEAP32 119

HERE 124

HS 126

HWND 127

I 132

I1HERE 133

I3HERE 134

IDT 136

INTOBJ 138

IOCTL 140

IRB 141

IRP 145

IRQ 148

KEVENT 152

KMUTEX 153

KSEM 154

LDT 155

LHEAP 158

LINES 160

LOCALS 161

M 163

MACRO 164

MAP32 169

MAPV86	171	TABLE	247
MOD	173	TASK	249
MSR	176	THREAD	250
NAME	178	TIMER	253
NET	181	TSS	255
NTCALL	185	TYPES	257
NTSTATUS	187	U	259
O	188	USB	261
OBJDIR	189	VER	265
OBJTAB	191	WATCH	266
OPINFO	193	WC	268
P	195	WD	270
PACKET	197	WF	272
PAGE	199	WHAT	274
PAGEIN	202	WIDTH	276
PAUSE	203	WINERROR	277
PCI	204	WL	278
PEEK	206	WMSG	280
PHYS	207	WR	281
POKE	209	WS	282
Print Screen Key	210	WT	284
PRN	211	WW	286
PROC	212	WX	288
PTR	215	X	290
QUERY	218	XFRAME	291
R	220	ZAP	294
RS	222		
S	223		
SERIAL	225		
SET	229		
SRC	235		
SS	236		
STACK	237		
SYM	241		
SYMLOC	244		
T	246		

Index

Preface



The *SoftICE Command Reference* explains the functionality of all SoftICE® commands. The various commands described in this reference operate on IA-32 hosts and targets with the following supported operating systems:

- ◆ Windows® NT
- ◆ Windows 2000
- ◆ Windows XP
- ◆ Windows Server 2003

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

- ◆ Command Name
- ◆ Command Type
 - ◇ Flow Control
 - ◇ Setting Breakpoints and Watches
 - ◇ Manipulating Breakpoints
 - ◇ Symbol and Source Commands
 - ◇ System Information
 - ◇ Display and Change Memory
 - ◇ I/O Port
 - ◇ Mode Control
 - ◇ Customization
 - ◇ Window Control
 - ◇ Miscellaneous
- ◆ Definition
- ◆ Syntax
- ◆ Use
- ◆ Output
- ◆ Command Example
- ◆ Related Commands

Accessibility

Prompted by federal legislation introduced in 1998 and Section 508 of the U.S. Rehabilitation Act enacted in 2001, Compuware launched an accessibility initiative to make its products accessible to all users, including people with disabilities. This initiative addresses the special needs of users with sight, hearing, cognitive, or mobility impairments.

Section 508 requires that all electronic and information technology developed, procured, maintained, or used by the U.S. Federal government be accessible to individuals with disabilities. To that end, the World Wide Web Consortium (W3C) Web Accessibility Initiative (WAI) has created a workable standard for online content.

Compuware supports this initiative by committing to make its applications and online help documentation comply with these standards. For more information, refer to:

- ◆ W3C Web Accessibility Initiative (WAI) at www.W3.org/WAI
- ◆ Section 508 Standards at www.section508.gov
- ◆ Microsoft Accessibility Technology for Everyone at www.microsoft.com/enable/

About Inline Editing

This SoftICE version has the ability to do **Inline Editing** of variables displayed in either the Locals Window or the Watch Window.

Usage

- 1 Navigate to the variable you wish to edit in either the Locals Window (WL) or the Watch Window (WW).
- 2 Use the hotkey sequence, **ALT-E**, to launch Inline Editing.
- 3 Edit your data.
- 4 Press either the **Enter** key to store your changes, or the **Escape** key to abort your changes.

Navigation Keys

The following keys are available for the Inline Editing feature:

- ◆ **Enter** - Stores your modifications.
- ◆ **Escape** - Aborts any changes.
- ◆ **Left/Right Arrow** - Changes your position within the edit field. Additionally, pressing either of these keys puts you into editing's Overtyping Mode.
- ◆ **Home** - Moves to the start of the field and additionally puts you into Overtyping Mode.

Current Limitations

- ◆ Editing of strings, floats, and bit fields is not possible in this release.
- ◆ You must be in the Locals/Watch Window before typing **Alt-E**.
- ◆ The Inline Editing hotkey is **not** remappable.

Notes: All input is done in hex.

When you enter Inline Editing, you will be placed in Overtyping Mode. This means that the information to the right of the edit field will be overwritten until you complete your edit. If you start typing in the edit field, the entire entry will be erased. This is the intended functionality.

SoftICE Commands



.

Type

Window Control

Definition

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 execution address (CS:EIP) visible and highlights it.

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

?

Type

Miscellaneous

Definition

Evaluate an expression.

Syntax

? *expression*

Use

The ? command only evaluates expressions. (Refer to *H* on page 114 for information about getting help.)

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

Type

Miscellaneous

Definition

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:

- ◆ 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.
- ◆ Enter mnemonic commands followed by a list of bytes and/or quoted strings separated by spaces or commas.
- ◆ Use the RETF mnemonic to represent 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]`

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, flush the memory image from the cache, or modify the module.

Example

Enter the command that instructs the assembler to assemble the instructions at an offset from the current code segment.

```
A CS:1234
```

As soon as you enter the command, 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.

ADDR

Type

System Information

Definition

Display or switch to an address context.

Syntax

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

process-name Name of any currently loaded process.

process-id Process ID: Each process has a unique ID.

KPEB Linear address of a 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.

If you switch to an address context that contains a Local Descriptor Table (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.

Under low memory conditions, Windows starts swapping data to disk, including inactive processes, parts of the page directory, and page tables. When this occurs, SoftICE might not be able to 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.

When entering a process-name for the ADDR command, the SoftICE TAB-AutoComplete feature can be used to finish a partially-typed name. Press TAB at any time while entering the process-name, and SoftICE will attempt to complete the name for you.

Output

For each context or process, ADDR shows the following information.

<i>CR3</i>	Physical address of the page directory that is placed into the CR3 register on a process switch to the process.
<i>LDT</i>	If the process has an LDT, this field has the linear base address of the LDT and the limit field for the LDT selector. Windows processes that have an LDT use the same LDT selector. For process switches, Windows sets the base and limit fields of this selector.
<i>KPEB</i>	Linear address of the Kernel Process Environment Block for the process.
<i>PID</i>	Process ID. Each process has a unique ID.
<i>NAME</i>	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
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
00387000		FD89E5E0	004E	4NT
*0121C000	E1172000:0187	FD88CCA0	0037	ntvdm
00030000		8013DD50	0000	Idle

See Also

PROC

ALTKEY

Type

Customization

Definition

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 default 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 pops up the SoftICE screen, use the following command.

```
ALTKEY alt z
```


ALTSCR

Type

Window Control

Definition

Display SoftICE on an alternate screen.

Syntax

ALTSCR [**mono** | **vga** | **off**]

<i>mono</i>	Redirects SoftICE output to an alternate monochrome monitor using a Hercules-compatible monochrome card. This mode displays 43 lines of text rather than the 25 lines displayed in text mode.
<i>vga</i>	Redirects SoftICE output to an alternate monitor using standard VGA mode.

Use

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

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

The default setting is ALTSCR mode OFF.

Note: 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 the chapter on “Customizing SoftICE” in the *Using SoftICE* document.

In the SoftICE program group, use Display Adapter Setup to select the monochrome monitor. SoftICE automatically starts 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.

Note: ALTSCR VGA Users - If you use an alternate screen in VGA mode, you must disable VGA on the graphics card that will be used to display Windows. You cannot use two cards that are in VGA mode at the same time. Consult the documentation for your graphics card to find the appropriate PCI slot or switches to set.

Example

The following command redirects screen output to the alternate monitor in standard VGA mode.

```
ALTSCR vga
```

ANSWER

Type

Customization

Definition

Auto-answer and redirect console to modem.

Syntax

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

<i>com-port</i>	If no comport is specified it uses COM1.
<i>baud-rate</i>	Baud-rate to use for modem communications. The default is 38400. The rates include 1200, 2400, 4800, 9600, 19200, 23040, 28800, 38400, 57600, and 115200.
<i>i=init</i>	Optional modem initialization string.

Use

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

You can place a default ANSWER initialization string in the SoftICE configuration settings. Refer to the chapter on “Customizing SoftICE” in the *Using SoftICE* document.

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 comport 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 38400 bps.

```
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

DIAL; NET; SERIAL

APC

Type

System Information

Definition

Display Asynchronous Procedure Calls (APC).

Syntax

APC [**address** | **TID** | **PID**]

address Location of an asynchronous procedure call.

TID Thread ID of thread you want to search for asynchronous procedure calls.

PID Process ID of process you want to search for asynchronous procedure calls.

Use

The APC command displays information about asynchronous procedure calls that are current in the system. If you enter APC with no parameters, SoftICE lists all asynchronous procedure calls queued for delivery in the currently running thread. You can also instruct SoftICE to walk through a specified thread or process.

Example

The following command displays information about an asynchronous procedure call.

```
APC
APC Object at 806D716C
    PKTHREAD 806E15E0
    APC Queue Flink806E1614 Blink 806E1614
Routines:
    Kernel 801A3B5Entoskrnl!NtVdmControl+130E
    Rundown 801A44DAntoskrnl!NtVdmControl+1C8A
    Normal 801A3CFAntoskrnl!NtVdmControl+14AA
    Normal Context 00000000
    Argument1 00000000Argument2 00000000
    ApcStateIndex 0
    ApcMode KernelMode
    In APC Queue
User mode APC Queue Empty
```

See Also

DPC

ATTACH

Type

Customization

Definition

Define a user environment that will be instantiated each time SoftICE pops up.

Syntax

```
ATTACH [ optional commandmodifiers ] [ idxnum ] addresscontextname  
[ tablename ]
```

<i>(command only)</i>	Displays a list of defined user environments.
<i>optional commandmodifiers</i>	(See the list of “Optional Command Modifiers” below.)
<i>idxnum</i>	Where <i>idxnum</i> is the entry in a list of attach definitions. This makes the attach definition <i>idxnum</i> the current active user environment. <i>Note:</i> If an index number is specified along with an address context name and/or a tablename, then the previously-defined entry will be overwritten.
<i>Addresscontextname</i>	Specifies the name of an address context to autoswitch to. <i>Note:</i> The address context does not exist. It is also possible to enter a name that will never exist. By doing so, you are using only the ATTACH command’s <i>tablename</i> feature.
<i>tablename</i>	Specifies the name of a table to autoswitch to upon popup. <i>Note:</i> The table name does not need to exist. If it cannot be found, the SoftICE normal auto table selection mechanism will be executed.

Optional Command Modifiers

The following optional comand modifiers are used with the ATTACH command:

“.detach” – Disassociates a user defined environment and allows auto switching. An alias for the detach command.

“.list” – Shows the list of “attach” definitions.

“.delete” – Deletes an entry from the list of “attach.”

“.clear” – Deletes an entry from the list of “attach.”

“.set idxnum” – Changes the active user environment to that defined by *idxnum*. An alias for the “attach idxnum” command.

“.create” – Creates a new entry in the “attach” definition table.

Use

When SoftICE pops up, it automatically locates the active address context and the table that is best suited for this address context. It then automatically changes to that context and table. Under most circumstances, this is the desired behavior. There are instances though when this becomes a hindrance. Namely, when debugging a user mode application or when there are many tables involved and you are only interested in one table. For example, while debugging the gdidemo program, you can debug a problem in a call to ntdll. In this case, you would want to have ntdll symbols loaded. However, by default, SoftICE will automatically switch to the gdidemo table on popup. If you use the “attach SoftICE to a user environment” feature, you can override this behavior.

Notes: At times it is desirable to have the attach functionality only change the address context or the table. This can be accomplished by providing a filler name in the addresscontextname and/or table name parameters.

It is not necessary for a particular address context or table to reside in memory at the point in time the record is created or used. If it is available, a switch will be made to the defined address/table. Otherwise SoftICE will default back to automatic selection.

Example

This first example adds gdidemo with a table of gdidemo. The * indicates that this is the active user environment.

```
:attach gdidemo ntdll
```

```
:attach
```

Idx	Context Name	Table Name
*0	gdidemo	gdidemo

The next two examples add two more entries. Note that when you create an entry, that entry becomes the active environment.

```
:attach.create csrss ntoskrnl
:attach.list
```

Idx	Context Name	Table Name
0	gdidemo	gdidemo
*1	csrss	ntoskrnl

This example creates an entry with an address context, 'junkname,' that does not exist. The purpose of this entry is to cause SoftICE to always choose the table 'ntdll' on any popup.

```
:attach.create csrss ntoskrnl
:attach.list
```

Idx	Context Name	Table Name
0	gdidemo	gdidemo
1	csrss	ntoskrnl
*2	junkname	ntdll

This example shows how to replace an entry in the table. Specify the index number and then the new address context name/table name.

```
:attach 0 gdidemo user32
:attach
```

Idx	Context Name	Table Name
0	gdidemo	user32
1	csrss	ntoskrnl
*2	junkname	ntdll

BC

Type

Manipulating Breakpoints

Definition

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 following command:

```
BC 1 5
```

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

BD

Type

Manipulating Breakpoints

Definition

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 following command.

```
BD 1 3
```

BE

Type

Manipulating Breakpoints

Definition

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 following command.

```
BE 3
```

BH

Type

Manipulating Breakpoints

Definition

List and 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:

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

SoftICE saves the last 32 breakpoints.

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 following command.

```
BH
```

BL

Type

Manipulating Breakpoints

Definition

List all breakpoints.

Syntax

BL [**-x** | **-a**]

-x	Display extended symbol format: table!symbol+offset.
-a	Do not convert addresses to symbols.

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.

Breakpoints have only two states: enabled or disabled (see BE and BD). The BL command displays an asterisk (*) next to the breakpoint index of any breakpoint that is disabled. If SoftICE was activated due to a breakpoint, the BL command will highlight that breakpoint.

BL will resolve breakpoint addresses to global symbols whenever possible. If a module is not loaded or the current address context is wrong, the address will be displayed.

Note: The address will also be displayed for any unresolved local symbols.

The **-x** option displays symbol names in table!symbol+offset form. The **-a** option suppresses the address-to-symbol conversion, and displays addresses only. These options have no effect on breakpoint types that do not use addresses.

Examples

The BL command is usually used without any parameters to display all the breakpoints that have been defined.

```
BL
```

```
00)    BPX main+21
01)    BPX _HalEnableSystemInterrupt
02)    *BPMD 0023:0012FED0 W DR0
03)    BPMD gStruct+4 W DR1
```

As you can see, four breakpoints are defined. Breakpoint 02 was set on a local variable, which is why it is displayed only as an address. It is also currently disabled, as indicated by the asterisk. If the same breakpoints were displayed with the `-x` option, the symbol table name would precede the symbols, as shown in the following example.

```
BL -x
```

```
00)    BPX TypesTest!main+21
01)    BPX hal!_HalEnableSystemInterrupt
02)    *BPMD 0023:0012FED0 W DR0
03)    BPMD TypesTest!gStruct+4 W DR1
```

BMSG

Type

Breakpoints

Definition

Set a breakpoint on one or more Windows messages.

Syntax

```
BMSG hWnd [L] [begin-msg [end-msg ]] [IF expression]  
[DO "command1;command2;..." ]
```

<i>hWnd</i>	Window handle value returned from <code>CreateWindow</code> or <code>CreateWindowEX</code> .
<i>L</i>	Logs messages to the SoftICE Command window.
<i>begin-msg</i>	Single Windows message or lower message number in a range of Windows messages. If you do not specify a range with an <i>end-msg</i> , only the <i>begin-msg</i> will cause a break. <i>Note:</i> For both <i>begin-msg</i> and <i>end-msg</i> , the message numbers can be specified either in hexadecimal or by using the actual ASCII names of the messages, for example, <code>WM_QUIT</code> .
<i>end-msg</i>	Higher message number in a range of Windows messages.
<i>IF expression</i>	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
<i>DO command</i>	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 the chapter on “Using Breakpoints” in the *Using SoftICE* document.

Use

The `BMSG` command is used to set breakpoints on a window’s message handler that will trigger when it receives 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 set to 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 can set multiple BMSG breakpoints on one window-handle, but 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. This range 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 SoftICE to log the breakpoint information to the SoftICE Command window when the breakpoint is triggered, instead of popping up. The message range on which the breakpoint triggers includes any message with a type value less than or equal to WM_CREATE. You can view the output from this breakpoint being triggered by popping into SoftICE and scrolling through the command buffer.

```
BMSG f4c L 0 wm_create
```

BPE

Type

Manipulating Breakpoints

Definition

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.

Caution: 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.

SoftICE expands any conditional expressions or breakpoint actions that are part 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 after editing, press the Enter key. To clear the breakpoint, press the Escape key.

BPINT

Type

Breakpoints

Definition

Set a breakpoint on an interrupt.

Syntax

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

int-number Interrupt number from 0 to 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 that 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 the chapter on "Using Breakpoints" in the *Using SoftICE* document.

Use

Use the BPINT command to pop up SoftICE whenever a specified processor exception, hardware interrupt, or software interrupt occurs. You can use the IF option to specify a conditional expression that limits the interrupts that trigger the breakpoint. You can use the DO option to specify SoftICE commands that 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 points to the first instruction of the interrupt or exception handler routine pointed to by the interrupt descriptor table (IDT.) If a software interrupt triggers the breakpoint, the instruction pointer (CS:EIP) points to the INT instruction that caused the breakpoint.

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

If a software interrupt occurs in a NTVDM, control is transferred to a Windows protected mode interrupt handler. This handler eventually calls down to the NTVDM's interrupt handler which is pointed to by the

NTVDM's Interrupt Vector Table). To go directly to the NTVDM's interrupt handler after the BPINT has occurred on a software interrupt instruction, use the following command:

```
G @ &0:(int-number*4)
```

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 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 system call breakpoints (software interrupt 2Eh) 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"
```

BPIO

Type

Breakpoints

Definition

Set a breakpoint on an I/O port access.

Syntax

```
BPIO port [verb] [debug register][IF expression] [DO  
"command1;command2;..." ]
```

port Byte or word value.

<i>verb</i>	Value	Description
	<i>R</i>	Read (IN).
	<i>W</i>	Write (OUT).
	<i>RW</i>	Reads and Writes.

<i>debug register</i>	Value
	<i>DR0</i>
	<i>DR1</i>
	<i>DR2</i>
	<i>DR3</i>

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 the chapter on “Using Breakpoints” in the *Using SoftICE* document.

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.

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. 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. 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 command defines 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.

```
BPIO 21 w if (al!=0xFF)
```

Note: In the previous example, 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.

```
BPIO 3fe r if ((al & 0xC0)==0xC0)
```

BPLOAD

Set a breakpoint on module load.

Syntax

```
BPLOAD [.t | .p] module-name [L | U | B] [IF expression] [DO "command1;command2;..."]
```

.t or .p SoftICE accepts the command modifiers (*.t* or *.p*) to limit the scope of a breakpoint for all breakpoint commands.

Value	Description
<i>.t</i>	Conditionally set the breakpoint to trigger in the active thread.
<i>.p</i>	Conditionally set the breakpoint to trigger in the active Process ID.

module-name Name of the module, or a search pattern with a wildcard, on which to set the breakpoint.

L Specifies to break on Module-Load.

U Specifies to break on Module-Unload.

B Specifies to break on both Module-Load and Module-Unload.

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 that execute when the breakpoint triggers.

Note: If an *L*, *U*, or *B* option is not specified, the breakpoint will only trigger on module loads.

Use

The BPLOAD command defines breakpoints that will be triggered when the specified module (or any module that matches the pattern if a wildcard is used) is loaded and/or unloaded.

The command modifier (*.t* or *.p*) limits the scope of the breakpoint. The *.t* modifier sets the breakpoint to trigger only in the currently active thread, while the *.p* modifier sets the breakpoint to trigger only in the active process.

The module-name can be an exact name or a pattern with one or more wildcard characters (*). If the module name contains spaces, it should be surrounded by double-quotes.

A module's extension is not considered when matching the module name to the pattern, so extensions should not be used in the module name. If an extension is given, it will be stripped off before setting the breakpoint.

A module-load breakpoint can trigger when the specified module is loaded, unloaded, or both. The parameter following the module name specifies when to break: if an **L** is used, the breakpoint will trigger on module load; if a **U** is used, module unload; and if a **B** is used, both load and unload will trigger the breakpoint. The default behavior, which is used if no parameter is given, is to break on module load only (**L**).

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 the chapter on "Using Breakpoints" in the *Using SoftICE* document.

Note: SoftICE does not support setting module-load breakpoints on 16-bit Windows modules.

Example

This example sets a breakpoint that will trigger when a module named `mymodule` loads or unloads.

```
BPLOAD mymodule B
```

This example sets a logging breakpoint, which will display a message to the SoftICE history window and update the hit count when `mymodule` loads.

```
BPLOAD mymodule L if BPLOG
```

This example will break when `mydll` loads if a variable in another module, `myexe`, is non-zero. When this breakpoint is triggered, it will execute the `MOD` command before popping up SoftICE.

```
BPLOAD mydll L IF myexe!BreakVar != 0 DO "mod;"
```

BPM

Type

Breakpoints

Definition

Set a breakpoint on memory access or execution.

Syntax

```
BPM[size][.t|.p] address [verb] [debug-reg]
[IF expression] [DO "command1;command2;..."]
```

size Size specifies the range covered by this breakpoint . For example, if you use double word, and the third byte of the dword is modified, a breakpoint occurs.

Value	Description
<i>B</i>	Byte.
<i>W</i>	Word.
<i>D</i>	Double Word.

.t or .p SoftICE accepts the command modifiers (*.t* or *.p*) to limit the scope of a breakpoint for all breakpoint commands.

Value	Description
<i>.t</i>	Conditionally set the breakpoint to trigger in the active thread.
<i>.p</i>	Conditionally set the breakpoint to trigger in the active Process ID.

verb

Value	Description
<i>R</i>	Read.
<i>W</i>	Write.
<i>RW</i>	Reads and Writes.
<i>X</i>	Execute.

debug-reg

Value

DR0

DR1

DR2

DR3

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 that 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 the chapter on “Using Breakpoints” in the *Using SoftICE* document.

Use

Use BPM breakpoints to have SoftICE pop up whenever certain types of accesses are made to memory locations. You can use the size and verb parameters to filter the accesses according to their type, and you can use the DO parameter to specify arbitrary SoftICE commands that execute 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 another 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, and the CS:EIP (Code Segment: Execution Index Pointer) points to the instruction in the code stream following the trapped instruction. For the X verb, SoftICE pops up before the instruction causing the breakpoint to trigger has executed, and the CS:EIP points to 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 you specify a verb of R, W or RW, *executing* an instruction at the specified address does not cause the breakpoint to occur.

If you specify a size of W (BPMW), it is a word-sized memory breakpoint, and you must specify an address that starts on a word boundary. If you specify a size of D (BPMD), the memory breakpoint is dword sized, and you must specify an address that starts on a double-word boundary.

Any breakpoint set on an address below 80000000h (2 GB) is address-context sensitive. That is, the breakpoint is triggered only when the address context in which the breakpoint was set is active. This includes WIN32 and 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.

```
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.

```
BPM CS:80204D20 x
```

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.

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

BPT

Type

Manipulating Breakpoints

Definition

Use a breakpoint description as a template.

Syntax

BPT [*.t* | *.p*] *breakpoint number*

.t or .p SoftICE accepts the command modifiers (*.t* or *.p*) to limit the scope of a breakpoint for all breakpoint commands.

Value	Description
<i>.t</i>	Conditionally set the breakpoint to trigger in the active thread.
<i>.p</i>	Conditionally set the breakpoint to trigger in the active Process ID.

breakpoint number 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 original breakpoint referenced by *breakpoint-index* is not altered. This command offers a quick way to modify the parameters of an existing breakpoint.

When SoftICE displays a breakpoint description, it expands any conditional expressions or 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 output by the command follows.

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

Press **Enter** to add the new breakpoint.

BPX

Type

Breakpoints

Key

F9

Definition

Set or clear a breakpoint on execution.

Syntax

```
BPX [.t | .p][address] [IF expression] [DO  
"command1;command2;..."]
```

.t or .p SoftICE accepts the command modifiers (*.t* or *.p*) to limit the scope of a breakpoint for all breakpoint commands.

Value	Description
<i>.t</i>	Conditionally set the breakpoint to trigger in the active thread.
<i>.p</i>	Conditionally set the breakpoint to trigger in the active Process ID.

address Linear address to set execution breakpoint.

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 that 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 the chapter on "Using Breakpoints" in the *Using SoftICE* document.

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

The command modifier (*.t* or *.p*) limits the scope of a breakpoint. The *.t* modifier conditionally sets the breakpoint to trigger in the active thread, while the *.p* modifier conditionally sets the breakpoint to trigger in the active Process ID.

You must set the *address* parameter to point to the first byte of the instruction opcode of the instruction on which you want to set the breakpoint. 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 at the address 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: You can 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 an NTVDM, 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. This method must be used 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.

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

SoftICE supports a maximum of 256 breakpoints when you use this command.

Any breakpoint set on an address below 80000000h (2 GB) is address-context sensitive. That is, they are only triggered when the context in which they were set is active. This includes WIN32 and WIN16 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 94).

```
BPX .1234
```

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 displays in the Data window.

```
BPX 80023455 do "db ds:dx"
```

See Also

FILE

BSTAT

Type

Breakpoints

Definition

Display statistics for one or more breakpoints.

Syntax

BSTAT [*breakpoint* #]

breakpoint # Breakpoint index number.

Use

Use BSTAT to display statistics on breakpoint hits, misses, and whether breakpoints popped up SoftICE or were logged. A breakpoint will be logged to the history buffer instead of popping up SoftICE if it has a conditional expression that uses the BPLOG expression macro. Since conditional expressions are evaluated when the breakpoint is triggered, it is possible to have evaluation run-time errors. For example, a virtual symbol may be referenced when that symbol has not been loaded, or a reference to a symbol may not be resolved because the memory is not present. In such cases, an error will be generated and noted in the Status and Scode fields under the Misc. column in the BSTAT output. For more information on breakpoint evaluation, refer to *Using SoftICE*.

Output

For each breakpoint, SoftICE displays the following information.

<i>BP #</i>	Breakpoint index, and if disabled, an * (asterisk).
Totals	
<i>Hits</i>	Total number of times SoftICE has evaluated the breakpoint.
<i>Breaks</i>	Total number of times the breakpoint has evaluated TRUE, and SoftICE has either popped up or logged the breakpoint.
<i>Popups</i>	Total number of times the breakpoint caused SoftICE to pop up.
<i>Logged</i>	Total number of times the breakpoint has been logged.
<i>Misses</i>	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 an error.

Current

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 or the breakpoint count has not expired.

Miscellaneous

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 example uses 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
```

C

Type

Miscellaneous

Definition

Compare two data blocks.

Syntax

```
C start-address L length start-address-2
```

start-address Start of first memory range.

L *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, SoftICE displays both bytes and 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

Type

System Information

Definition

Display information on Windows classes.

Syntax

```
CLASS [-x][process | thread | module | class-name]
```

<i>-x</i>	Display complete Windows internal CLASS data structure, expanding appropriate fields into more meaningful forms.
<i>process</i>	Process name, process ID, or process handle.
<i>thread</i>	Thread ID or thread address (KTEB).
<i>module</i>	Module name or module handle.
<i>class-name</i>	Name of a registered class window.

Use

The architecture of class information is process specific and the operating system creates different lists for global and private classes. 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:

<i>Class Name</i>	Name that was passed when the class was registered. If no name was passed, the atom displays.
<i>Handle</i>	Offset of a data structure within USER. Refers to windows of this class.
<i>Module</i>	Module that has registered this window class.
<i>Wndw Proc</i>	Address of the window procedure for this window class.
<i>Styles</i>	Bitmask of flags specified when the class was registered.

Example

The following example uses the CLASS command to display classes registered by the Microsoft WORD module.

```
CLASS winword
```

Class-Name	Handle	Module	Wndw Proc	Styles
File Open WNet Window	A032B4E0	Mso9	30B525BD	00000000
CLIPBRDWNDCLASS	A0374978	ole32	77A6C469	00000000
OleDdeWndCLASS	A0373AE8	ole32	77A769D0	00000000
sa_sdm_Microsoft Word 9.	A0373948	WINWORD	309DCE01	00000028

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

Type

Window Control

Key

Alt-F5

Definition

Clear the Command window.

Syntax

CLS

Use

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

CODE

Type

Customization

Definition

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.
- ◆ Use CODE with no parameters to display 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

Type

Customization

Definition

Display or set the screen colors.

Syntax

`COLOR [normal bold reverse help line] | [RESET]`

<i>normal</i>	Foreground/background attribute that displays normal text. Default = 07h gray on black.
<i>bold</i>	Foreground/background attribute that displays bold text. Default = 0Fh white on black.
<i>reverse</i>	Foreground/background attribute that displays reverse video text. Default = 71h blue on gray.
<i>help</i>	Foreground/background attribute that displays the help line underneath the Command window. Default = 30h black on cyan.
<i>line</i>	Foreground/background attribute that displays the horizontal lines between the SoftICE windows. Default = 02h green on black.
<i>RESET</i>	Reset all colors to their default values.

Use

Use the COLOR command to customize the SoftICE screen colors on a color monitor. Each of the five colors can be specified in either hexadecimal or textual format. If a hexadecimal format is used, the foreground color is specified in bits 0-3, while the background color is specified in bits 4-7. This is identical to the standard CGA attribute format in which there are 16 foreground colors and 8 background colors.

If text is used, the colors are specified as *foreground+background* (e.g., *gray+blue* for gray text on a blue background). If the background color is omitted, the current background color for normal text will be used as a default. No spaces are allowed within any color argument.

Note: Hexadecimal colors can be combined with textual format colors on the same command line.

The 16 color names to use when designating colors via textual format, and the corresponding hexadecimal codes, are listed in the following table.

Code	Color	Code	Color
0	black	A	lightgreen
1	blue	B	lightcyan
2	green	C	lightred
3	cyan	D	lightmagenta
4	red	E	yellow
5	magenta	F	white
6	brown		
7	gray		
8	darkgray		
9	lightblue		

Example

The command below makes the color assignments that are listed following the command.

```
COLOR 7 f 71 30 2
```

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

Here is the same command using textual colors.

```
COLOR gray+black white+black blue+gray black+cyan green+black
```

CPU

Type

System Information

Definition

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).

If your PC contains a multi-processor motherboard that uses an I/O Advanced Program Interrupt Controller (APIC) as an interrupt controller, the CPU command displays the CPU local registers and the I/O APIC information.

Example

The following example lists the sample output from the CPU command 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=FFF0FF0
DR7=00000400
EFL=00000246 PF ZF IF IOPL=0
```

The following example lists the sample output from the CPU command 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 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

CSIP

Type

Breakpoints

Definition

Set the Code Segment:Execution Index Pointer (CS:EIP) memory range qualifier for all breakpoints (16-bit code and modules).

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

Syntax

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

<i>off</i>	Turns off CSIP checking.
<i>not</i>	Breakpoint only occurs if the CS:EIP is outside the specified range or module.
<i>start-address</i>	Beginning of memory range.
<i>end-address</i>	End of memory range.
<i>module-name</i>	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

The CSIP command still works for 16-bit code and modules. CSIP qualifies breakpoints so that the code that triggers 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 the instruction pointer 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.

Since 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 you do not specify parameters, the current memory range displays.

Remember that this command is obsolete for 32-bit code. Use conditional expressions to achieve this functionality.

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

Type

Display/Change Memory

Definition

Display memory.

Syntax

`D[size] [-p] [address [l length]]`

size

Value	Description
<i>B</i>	Byte (8-bits)
<i>W</i>	Word (16-bits)
<i>D</i>	Double Word (32-bits)
<i>S</i>	Short Real (32-bits)
<i>L</i>	Long Real (64-bits)
<i>T</i>	10-Byte Real (80-bits)

-p

Indicates that *address* is a physical address rather than a virtual one.

address

Starting address of the memory you want to display.

l length

Displays *length* number of bytes to the Command window.

Use

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

SoftICE displays the memory contents in the format you specify in the *size* parameter. If you do not specify a size, SoftICE uses the last size specified. For the byte, word, and double word hexadecimal formats, the ASCII representation is displayed.

The D command displays data either in the active data window, or if there is no data window open, the command window. If the data is displayed in the command window, the D command will display 8 lines of data by default.

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.

The *address* parameter may be any virtual address in the system. Selector overrides are allowed; if you do not specify a selector as part of the address, SoftICE will use the last selector specified (in practice, the selector can usually be ignored in a flat address environment). If you need to display a physical address, you can use the `-p` switch. Physical address space is flat, not segmented, so when displaying physical addresses the `D` command will show `PHYS:` in place of the selector.

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 would result in a fractional row, SoftICE rounds up. This form of the `D` command is useful when dumping large amounts of data to the command window for the purpose of writing it to a log file.

For floating point values, numbers 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	+/-

Example

Displays the memory starting at address `DS:F1479000` in byte format and in ASCII format:.

```
DB ds:F1479000
```

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
```

The following command displays memory starting at physical address 1158D000 in word format. Note that the physical addresses are flat; selector values are not used.

```
DW -p 1158D000
```

See Also

DATA; WD

DATA

Type

Window Control

Definition

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 the chapter on “Customizing SoftICE” in the *Using SoftICE* document.

Example

The following command changes the visible Data window to Data window number 3.

```
DATA 3
```

DETACH

Type

Customization

Definition

Disassociates a user environment from the current environment.

Syntax

DETACH

Use

When a user has "attach"ed to an environment upon popup, the debugging environment will be overridden by that defined with "attach." When you issue the "detach" command, you allow SoftICE to go back into automatic mode and choose the environment that is appropriate for SoftICE popups. This means that address contexts and tables will be automatically chosen at the time of the SoftICE popup.

Example

The following command disassociates a user environment from the current environment.

```
DETACH
```

See Also

ATTACH

DEVICE

Type

System Information

Definition

Display information on Windows device objects.

Syntax

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

device-name Object directory name of the device.

pdevice-object Object address of the device.

Use

The **DEVICE** command displays information on Windows 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 entire object tree. 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:

<i>RefCnt</i>	Device object's reference count.
<i>DrvObj</i>	Pointer to the driver object that owns the device object.
<i>NextDev</i>	Pointer to the next device object on the linked list of device objects that were created by the same driver.
<i>AttDev</i>	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.

<i>CurIrp</i>	Pointer to the IRP currently being serviced for the device object by the device object's driver.
<i>DevExten</i>	Pointer to device driver-defined device object extension data structure.
<i>Name</i>	Name of the device, if it has one.

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

<i>Flags</i>	Definition of the device object's attributes such as whether I/O performed on the device is buffered or not.
<i>Vpb</i>	Pointer to the device's associated volume parameter block.
<i>Device Type</i>	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

The following 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			

DEVNODE

Type

System Information

Definition

Display information about device nodes.

Syntax

```
DEVNODE [-c | -r | -t | -x | -p] address | service-name
```

<i>-c</i>	Dump the list of children for a device node at a specific address.
<i>-r</i>	Dump the raw resource list for a device node at a specific address.
<i>-t</i>	Dump the translated resource list for a device node at a specific address.
<i>-x</i>	Dump extended information for a device node at a specific address.
<i>-p</i>	Dump the parent tree for the device node.
<i>address</i>	Dump information for a device node at a specific address.
<i>service-name</i>	Dump information for all device nodes with a specific service name.

Use

The DEVNODE command displays information about device nodes (PnP manager structure). If you use DEVNODE without any parameters, SoftICE dumps the root device node. If you specify an address, SoftICE dumps information about the device node at that address. If you specify a service name, or a partial name, SoftICE dumps information about all device nodes whose service names match the service name or partial name entered. Using the various flags also allows you to control the dumping of child lists, resource lists, and extended information.

Examples

The following example shows the DEVNODE command dumping the child list for a device node at a specific address.

```
DEVNODE -c 86f956f0
DEVICE_NODE
Address      Child      LastChild  InstancePath      ServiceName
-----
86dc2010    00000000    00000000    Root\MEDIA\MS_MMVID    audstub
86dc2560    00000000    00000000    Root\MS_PSCHEMP\0002    PSched
...
86f94b30    00000000    00000000    Root\dmio\0000        dmio
86f950c0    86f8e740    86f8e740    Root\ACPI_HAL\0000
```

The following example shows the DEVNODE command dumping the parent tree for a device node at a specific address.

```
DEVNODE -p 80e3d9e0
PARENT TREE
Address      Child      LastChild  InstancePath      ServiceName
-----
80e7a4c8    80eb3ee8    80e764c8    HTREE\ROOT\0
80eb3ee8    80eafc00    80eafc00    Root\ACPI_HAL\0000
...
```

See Also

DEVICE, DRIVER

DEX

Type

Customization

Definition

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.

expression Data expression to assign to 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 59.)

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 to by the public variable `PointerVariable`.

```
DEX 1 @pointervariable
```

See Also

DATA

DIAL

Type

Customization

Definition

Redirect console to modem.

Syntax

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

<i>com-port</i>	If no comport is specified, the default is COM1.
<i>baud-rate</i>	Baud-rate to use for modem communications. The default is 38400. The rates you can specify are 1200, 2400, 4800, 9600, 19200, 23040, 28800, 38400, 57000, and 115000.
<i>i=init-string</i>	Optional modem initialization string.
<i>p=number</i>	Telephone number.

Use

The DIAL command initiates a call to a remote machine via a modem. The remote machine must be running SIREMOTE.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. For a detailed explanation of this procedure, refer to “Using SoftICE with a Modem” in the *Using SoftICE* document.

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 the chapter on “Customizing SoftICE” in the *Using SoftICE* document.

Example

The following is an example of the DIAL command:

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

This command tells SoftICE to first initialize the modem on com-port 2 at a baud rate of 19200 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 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 this Dial initialization string in place, SoftICE always initializes the modem specified in DIAL commands with “ATX0”, unless the DIAL command explicitly specifies a different initialization string.

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

See Also

ANSWER; NET; SERIAL

DPC

Type

System Information

Definition

Display Deferred Procedure Calls.

Syntax

DPC [*address*]

address Location of a delayed procedure call.

Use

The DPC command displays information about deferred procedure calls that are current in the system. If you enter DPC without parameters, SoftICE list all delayed procedure call that are queued for delivery in the system.

Example

The following command displays a listing of all deferred procedure calls current in the system.

```
DPC
```

See Also

APC

DRIVER

Type

System Information

Definition

Display Windows driver information.

Syntax

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

driver-name Object directory name of the driver.

pdriver-object Object address of the driver.

Use

The DRIVER command displays information on Windows 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 entire object tree. 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:

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

This example shows the output of the DRIVER command with no parameters. Notice that 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 example shows 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
-------	------	---------	-------	---------	----------	---------

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

Type

Display/Change Memory

Definition

Edit memory.

Syntax

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

size

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

address

Starting address in memory.

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 the 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.

```
EB ds:1000 3.1415
```

EC

Type

Window Control

Key

F6

Definition

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

In source mode, you can scroll the Code window from the Command window using the CTRL key with one of cursor keys described above. In this mode, 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.

ERESOURCE

Type

System Information

Definition

Display information about the synchronization resources contained in ExpSystemResourceList.

Syntax

```
ERESOURCE [ -a | -c | -w | address ]
```

<i>-a</i>	Display resources that are actively held by any thread.
<i>-c</i>	Display resources that are or have been under contention (where contention count > 0).
<i>-w</i>	Display resources that have threads currently waiting on them.
<i>address</i>	Address of an ERESOURCE structure.

Use

This command displays the ERESOURCE structure, a list of the threads that currently own the ERESOURCE, and a list of the threads that are waiting on the ERESOURCE.

When you do not specify an address, SoftICE displays summary information about every ERESOURCE structure in ExpSystemResourceList.

Example

Enter the following command to display a list of the active resources on your system.

```
ERESOURCE -a
```

You can enter the following command to get extended information about a specific ERESOURCE structure, including thread contentions and threads waiting on the ERESOURCE.

```
ERESOURCE address
```

You can use the information you get from the commands above in combination with the following command to help find deadlocks.

```
ERESOURCE -w
```

See Also

KEVENT; KSEM; THREAD

EVENT

Type

System Information

Definition

Displays BoundsChecker® events.

Syntax

EVENT [-? | -a | -d | -lx | -nd | -o | -pd | -r | -s | -t | -x]
[start-event-index [L event-count]]

- ? Displays descriptions of the supported command switches.
- a Turns API return display on or off. The default setting is on. When this is off, SoftICE does not display API return events.
- d Lists all drivers currently being monitored by BoundsChecker. Type **Event-d** at the SoftICE Command Prompt to see a listing of these drivers and the total count.
- lx Specifies the stack-checking level (0x40 - 0x4000). The default setting is 0x800.
- nd Specifies the nesting depth used to display events. Legal values are 0 to 32 (decimal format). The default nesting level is 10. If events nest past the specified nesting depth, SoftICE does not display them as indented.
- o Turns event logging on or off. The default setting is on.
- pd Specifies the SoftICE pop-up level for BoundsChecker events. The default setting is 0.

Value	Meaning
0	SoftICE does not pop up on BoundsChecker events.
1	SoftICE pops up on errors only.
2	SoftICE pops up on all errors and warnings.

- r Clears the event buffer.
- s Displays the current status of event viewing and logging. The number of logged events is the total that have been trapped since the system was started. It is displayed in decimal format.
- t Turns display of thread switches on or off. The default setting is on. When this option is on and event n-1 is in a different thread than

	event n, SoftICE displays event n in reverse video indicating a thread switch has occurred. When this option is off, SoftICE does not display thread switches.
-x	Displays all events with their parameters, as well as general summary information for each event, including elapsed time, current thread and current IRQ. If you do not specify this switch, SoftICE displays a single summary line for each event.
<i>start-event-index</i>	Displays events starting at the specified event index.
<i>L event-count</i>	Displays the logged events in the Command window, starting from the specified start-event-index for a length of event-count events. If you do not specify a length, SoftICE displays the events in a scrollable window starting from start-event-index (if one is specified).

Use

Use the **EVENT** command to display information about BoundsChecker events. You can display event information in the Event window or in the Command window.

Viewing Events in the Event Window

You can specify whether SoftICE displays the events in the Event window with summary or detail information. While the Event window is open, you can use F1 to expand or collapse all events. You can place the cursor on a line and double-click or press Enter to expand or collapse a single event.

The Event window supports the following keys.

Key	Description
<i>Enter</i>	Toggles the display state of the event at the current cursor position between summary information and detail information.
<i>Esc</i>	Closes the Event window. When you re-open the Event window, SoftICE preserves the previous window state (i.e. current event, expansion state, and filters are the same).
<i>PageUp</i>	Scrolls the screen up one page.
<i>PageDown</i>	Scrolls the screen down one page.
<i>Up Arrow</i>	Moves cursor up one line. If on the top line, it scrolls the window up one line.
<i>Down Arrow</i>	Moves cursor down one line. If on bottom line, it scrolls window down one line.

Key	Description (Continued)
<i>Shift-Left Arrow</i>	Scrolls the window left one column.
<i>Shift-Right Arrow</i>	Scrolls the window right one column.
<i>Home</i>	Moves the cursor to the top row. If the cursor is already on the top row, starts display at the first event.
<i>End</i>	Moves the cursor to the bottom row. If the cursor is already on the bottom, starts display at the last event.
<i>*</i>	Undoes the last Home or End operation.
<i>F1</i>	Toggles the display state of all events between summary information and detail information.
<i>F2</i>	Displays the Event filtering dialog.
<i>F3</i>	Displays the Parameter filtering dialog.
<i>F4</i>	Displays error events only.
<i>F</i>	Closes the Event window and returns focus to the Command window. Use this key if you want to use other SoftICE commands on data that is displayed in the Event window. If you bring up the Event window again, SoftICE preserves the previous window state (i.e. current top event, expansion state, and filters are the same).
<i>R</i>	Toggles the display state of API returns between showing all API returns and showing no API returns.
<i>T</i>	Toggles the highlighting of thread switches. Thread switches are indicated by displaying the summary line of the first event in the new thread in reverse video.
<i>E</i>	Toggles the highlighting of errors on API returns. SoftICE displays the summary line of API return errors in bold.
<i>S</i>	Displays the event at the current cursor position at the top of the Event window.
<i>N</i>	Finds the next event that matches the search criteria selected with the right mouse button.
<i>P</i>	Finds the previous event that matches the search criteria selected with the right mouse button.
<i>0 - 7</i>	Filters events by CPU number on SMP machines. Each key acts as a toggle for displaying all events that occurred on a specific CPU. These keys also appear as buttons on the top line of the Event window.

Viewing Events in the Command Window

In the Command window, SoftICE can display any number of events starting from any specific event index. SoftICE can display the events

with summary or detail information. The summary display includes only a single line for each event. The detail display includes the summary information, as well as all event parameters. You can use the EVENT command switches to customize the display output.

It is useful to view events in the Command window when you want to view a small group of functions, or when you want to save the event data to a SoftICE History file. A SoftICE History file contains current contents of the SoftICE history buffer. You can use the scroll bars in the Command window to view the contents of the SoftICE history buffer. For more information, see Chapter 2, “Using BoundsChecker Driver Edition,” in the *Using DriverStudio Tools* document.

Example

Enter the following command at the command prompt to display events in the Event window.

```
EVENT
```

When you do not specify *start-event-index* or *event-count*, SoftICE displays the Event window in place of the Command window. You can use this command with one of the EVENT command switches or with a *start-event-index* to customize the display.

Enter the following command at the command prompt to display events in the Command window starting at event *start-event-index* for a length of *event-count* events.

```
EVENT start-event-index event-count
```

See Also

EVMEM

EVMEM

Type

System Information

Definition

Display information about BoundsChecker memory events.

Syntax

EVMEM [-? | -d | -t | -s | -p | -o | -e] [tag | driver-name | pool-type]

-?	Displays descriptions of the supported command switches.																
-d	Sorts the output by driver name.																
-t	Sorts the output by tag.																
-s	Sorts the output by size.																
-p	Sorts the output by pool type.																
-o	Displays overview information.																
-e	Displays only error events.																
tag	Displays only memory events that were allocated with that specific tag. Tags are 4 byte ASCII strings that are passed to the ExAllocatePoolWithTag API.																
driver-name	Displays memory events for only the specified driver.																
pool-type	Displays only memory events allocated out of that specific pool. The following values are valid. <table><tr><td>NPP</td><td>Non-paged pool</td></tr><tr><td>PP</td><td>Pageable pool</td></tr><tr><td>NPPMS</td><td>Non-paged pool, must succeed</td></tr><tr><td>NPPCA</td><td>Non-paged pool cache aligned</td></tr><tr><td>PPCA</td><td>Pageable pool cache aligned</td></tr><tr><td>NPPCAMS</td><td>Non-paged pool cache aligned, must succeed</td></tr><tr><td>MMC</td><td>Allocated by MMAllocateContiguousMemory API</td></tr><tr><td>MMNC</td><td>Allocated by MMAllocateNonCachedMemory API</td></tr></table>	NPP	Non-paged pool	PP	Pageable pool	NPPMS	Non-paged pool, must succeed	NPPCA	Non-paged pool cache aligned	PPCA	Pageable pool cache aligned	NPPCAMS	Non-paged pool cache aligned, must succeed	MMC	Allocated by MMAllocateContiguousMemory API	MMNC	Allocated by MMAllocateNonCachedMemory API
NPP	Non-paged pool																
PP	Pageable pool																
NPPMS	Non-paged pool, must succeed																
NPPCA	Non-paged pool cache aligned																
PPCA	Pageable pool cache aligned																
NPPCAMS	Non-paged pool cache aligned, must succeed																
MMC	Allocated by MMAllocateContiguousMemory API																
MMNC	Allocated by MMAllocateNonCachedMemory API																

Use

Use the EVMEM command to display information about BoundsChecker memory events in the Command window. To display information about all types of events, use the EVENT command. For more information, see Chapter 2, “Using BoundsChecker Driver Edition,” in the *Using DriverStudio Tools* document.

Example

Enter the following command at the command prompt to display memory events in the Command window.

```
EVMEM
```

You can use the EVMEM command switches to customize the display, including sorting the output and displaying additional information.

Enter the following command at the command prompt to display events in the Command window for driver-name:

```
EVMEM driver-name
```

See Also

EVENT

EVRES

Type

System Information

Definition

Displays resources collected by the BoundsChecker driver, BCHKD.SYS.

Syntax

EVRES [Process-Type | Object-Type | Driver-Type]

Process-Type A Process-Type is a process name, a PID, or a PCB address. If one is specified, only objects created in that process will be displayed. Use this version of the command to display only objects created in the system process:

EVRES system

Object-Type An Object-Type is one of the following:

- KEY
- DIRECTORY
- INTERRUPT
- FILE
- SECTION
- EVENT

These refer to the types of objects collected by BCHKD. If one is specified, only the objects of that type will be displayed. Use this version of the command to display interrupt objects:

EVRES interrupt

Driver-Type A Driver-Type is a driver name. If one is specified, only resources created by that driver will be displayed. Use this version of the command to display resources created by the netbios driver:

EVRES netbios

Note: If no parameters are entered, all resources will be displayed.

For each captured resource, the following information will be displayed:

- ◆ **Handle** – This is the object handle of the resource. For interrupt objects, it is the address of the interrupt object structure.
- ◆ **Process** – This is the process name and process id where the resource was created.
- ◆ **Obj Type** – This is one of the object types listed above.

- ◆ **Name** – This is the resource name. For interrupt objects, this is the interrupt vector number and the interrupt service routine address.
- ◆ **EIP1** – This is the address in the driver that created the resource. If a symbolic name is available, it will be displayed; otherwise, the address and the driver name plus an offset will be displayed.
- ◆ **EIP2** – This is the second level of return address on the stack. If a symbolic name is available, it will be displayed; otherwise, the address and the driver name plus an offset will be displayed.

Use

Use the EVRES command to display resources collected by the BoundsChecker driver BCHKD.SYS. For more information, see Chapter 2, “Using BoundsChecker Driver Edition,” in the *Using DriverStudio Tools* document.

Example

The following is a sample of the output of an EVRES command:

Handle	Process(PID)	Obj Type	Name
8147A768	System(08)	INTERRUPT	Vec:51 ISR:ED0907A5 EIP1: ED092F20 serial!PAGESRP0+0720 EIP2: 00000000
8147AA28	System(08)	INTERRUPT	Vec:A2 ISR:ED0907A5 EIP1: ED092F20 serial!PAGESRP0+0720 EIP2: 00000000
8147B008	System(08)	INTERRUPT	Vec:52 ISR:ED086D10 EIP1: ED083526 i8042prt!PAGE+0406 EIP2: ED0844F1 i8042prt!PAGE+13D1
8155C628	System(08)	INTERRUPT	Vec:B3 ISR:ED0810CC EIP1: ED08360D i8042prt!PAGE+04ED EIP2: ED0844DA i8042prt!PAGE+13BA
8155C008	System(08)	INTERRUPT	Vec:93 ISR:ED3124BC EIP1: ED316D70 uhcd!PAGE+0B50 EIP2: ED310FB3 uhcd!.text+0CD3
81579008	System(08)	INTERRUPT	Vec:83 ISR:BFECB591 EIP1: BFEC360B NDIS!NdisInitializeInterrupt+0179 EIP2: BFEC348B NDIS!NdisMRegisterInterrupt+0035

818AB008	System(08)	INTERRUPT Vec:92 ISR:BFF27E28 EIP1: BFF300C4 atapi!PAGE+0AA4 EIP2: BFF2FF37 atapi!PAGE+0917
818AB408	System(08)	INTERRUPT Vec:92 ISR:BFF27E28 EIP1: BFF300C4 atapi!PAGE+0AA4 EIP2: BFF2FF37 atapi!PAGE+0917
818ABC68	System(08)	INTERRUPT Vec:72 ISR:BFF27E28 EIP1: BFF300C4 atapi!PAGE+0AA4 EIP2: BFF2FF37 atapi!PAGE+0917
818AB008	System(08)	INTERRUPT Vec:71 ISR:BFF27E28 EIP1: BFF300C4 atapi!PAGE+0AA4 EIP2: BFF2FF37 atapi!PAGE+0917
814D5008	System(08)	INTERRUPT Vec:B1 ISR:BFF7F44Av EIP1: BFF8FF8E ACPI!PAGE+08CE EIP2: BFF97403 ACPI!PAGE+7D43
Total Resource Objects:		
10		

See Also

EVENT; EVMEM

EXP

Type

Symbol/Source

Definition

Display export symbols for a module.

Syntax

```
EXP [[module!*][symbol*]] | [!]
```

<i>module</i>	A valid module name (or partial module name) to use as a search pattern.
<i>symbol</i>	A valid symbol name (or partial symbol name).
!	Display loaded export tables.

Note: The EXP command supports regular expression searches.

Use

Use the EXP command to display exported symbol addresses. For exports to be displayed, you must have previously loaded exports for the module using Loader32 or NMSym. SoftICE automatically loads exports for KERNEL32, USER32, GDI32, NTOSKRNL, and HAL. The EXP command will work with both 16- and 32-bit modules.

If you do not specify a module name, EXP will search all the loaded export tables for symbols matching the one you specify. If you do not specify any parameter, all symbols are displayed.

The command EXP ! can be used to display a list of the loaded export tables.

The EXP command can work with partial module and symbol names containing wildcards, and also with regular expressions, for more advanced searches. For simple searches there are two wildcards, asterisk (*), which will match any sequence of characters, and comma (,), which will match any single character.

To use a regular expression to search for symbols, the search patterns must be enclosed in slash (/) characters. If the search patterns contain spaces, the whole expression, including slashes, must also be enclosed in double-quotes ("). There may be two separate search patterns, one for the module, and one for the symbol name, and either or both may be regular expressions. The full syntax of regular expressions is beyond the scope of this manual, but some examples are given below, and references can be found on the Internet.

Search expressions are case-insensitive by default. Use the SET CASESENSITIVE command to toggle case sensitivity on and off.

Example

The following example shows all exported symbols containing the string atom.

```
EXP *atom*
```

The following example uses a regular expression to display all exported symbols containing the string atom.

```
EXP /atom/
```

The following two examples will both find all symbols in KERNEL32 that begin with the string check. The caret (^) at the beginning of the regular expression in the second example specifies that the next character must match the beginning of the symbol. Without the caret this pattern would match any symbol containing the string check at any position within the name.

```
EXP kernel32!check*  
EXP kernel32!/^check/
```

See Also

SYMLOC; TABLE

F

Type

Miscellaneous

Definition

Fill memory with data.

Syntax

F *address l length data-list*

<i>address</i>	Starting address at which to begin filling memory.
<i>l length</i>	Length in bytes.
<i>data-list</i>	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 length specified by the L parameter. If the data-list length is less than the specified length, the data-list is repeated as many times as necessary.

Example

The following 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

Type

Mode Control

Definition

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

The following example turns off fault trapping in SoftICE.

```
FAULTS off
```

See Also

SET

FIBER

Type

System Information

Definition

Dump a fiber data structure.

Syntax

FIBER [*address*]

address Thread address. If you do not specify an address, FIBER dumps the fiber data using the address associated with the current thread.

Use

Use the FIBER command to dump a fiber data structure as 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

Type

Symbol/Source

Definition

Change or display the current source file.

Syntax

FILE `[[*]file-name]`

*** Display all files in the current symbol table.

file-name Name of file to make current source file.

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 (*) character, 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.

The SoftICE TAB-AutoCompletion feature can be used to finish partially-typed filenames. Press TAB at any time while entering a filename, and SoftICE will attempt to finish the name for you.

When you specify a file name in the FILE command, SoftICE switches address contexts if the current symbol table has an associated address context.

Example

Assuming `main.c` is loaded with the Symbol Loader, the following command displays the file in the Code window starting with line 1.

```
FILE main.c
```

FILTER

Creates or deletes display filters for the SoftICE history window.

Syntax

FILTER [-d] [*string* | /*regex*/[i]] [*color*] [DO *action*]

<i>-d</i>	This option specifies that a filter is to be deleted. Filters are deleted by number; use the FILTER command by itself to see a list of defined filters.
<i>string</i>	The text to compare the display output against. FILTER always does substring matches on a display line. If the string contains spaces it must be delimited by double-quotes (""). Substring matches in filters are always case-insensitive.
/ <i>regex</i> /	A regular expression to compare the display output against. Regular expressions must always be delimited by '/' characters; if the regular expression contains spaces the whole thing must be enclosed in double-quotes. Regular expression matches are case-insensitive by default, but if the global CaseSensitive setting is TRUE, the second '/' delimiter may be followed by an 'i' for a case-insensitive search.
<i>color</i>	The color to apply to the line when text matches the filter. Colors can be specified numerically or textually.
DO <i>action</i>	A list of SoftICE commands to execute if the filter matches display text. The action must be delimited by double-quotes, and can include multiple commands separated by semicolons.

Use

The FILTER command creates or deletes display filters. Display filters are used to selectively color lines in the history window, and optionally to execute a SoftICE command whenever a line to be displayed in the history window matches the filter. A display filter consists of a search pattern, which can be either a simple string or a regular expression; a color combination to be used when displaying the matching line; and an optional action to be executed when a line matches.

The SoftICE history window displays both SoftICE output and any text output from drivers using the DbgPrint API. Display filters are applied to all text that is displayed after the filter is created.

Filters can use either a simple substring match, or the more powerful regular expression syntax. If simple strings are used, no wildcards are supported, and the comparison is always case-insensitive. When regular expressions are used, case sensitivity is controlled by the global CaseSensitive flag (see the [SET](#) command). When CaseSensitive is on, the second '/' delimiter of the regular expression may be followed by an 'i' to specify a case-insensitive search. Regular expression syntax is beyond the scope of this manual, but is widely available on the Internet.

The filter color consists of a foreground and background, and can be specified in hex or textually. If hex is used, the foreground color is specified in bits 0-3, and the background in bits 4-7. For textual colors, the format is "<foreground>+<background>". The background color can be omitted; if it is omitted, then the background color for normal text will be used.

The 16 color names and their codes are listed in the following table.

Code	Color	Code	Color
0	black	A	lightgreen
1	blue	B	lightcyan
2	green	C	lightred
3	cyan	D	lightmagenta
4	red	E	yellow
5	magenta	F	white
6	brown		
7	gray		
8	darkgray		
9	lightblue		

Filters may also have actions associated with them. Actions are specified using the same syntax as breakpoint actions: DO "cmd; cmd;". Filter actions can only be executed when SoftICE is not already popped up. If SoftICE is popped up and displays text that matches a filter with an associated action, the action will simply be ignored. This applies to text displayed by SoftICE itself as well. SoftICE will not execute actions from filters when they apply to its own output.

Filter actions also automatically cause SoftICE to popup when they are executed. This means that filters can be used much like breakpoints. If you do not want SoftICE to popup on an action, you can insert an "x;" as the last command in your action, which will exit SoftICE as soon as the action is executed. (The trailing semicolon is required).

Used without parameters, FILTER will display a numbered list of the defined filters. To delete a filter, the syntax is “FILTER -d <filter number>”.

If multiple filters would apply to the same line, they are applied in order. This means that the highest numbered filter that matches a line will set the color for that line.

Filters are saved in the SoftICE `winice.cfg` file when Windows is shut down or restarted, so they will persist from one debugging session to the next.

Example

First a trivial example. Imagine that you want to quickly see all the SoftICE commands that operate on the data window:

```
:FILTER “data window” red+white  
:h
```

The help command will execute, and all SoftICE commands that contain the text “data window” anywhere in their description will display as red text on a white background, making them easier to spot.

If you have written a driver that prefixes all its `DbgPrint` output with a certain string, you can easily set up a filter to display all your driver's output in a different color:

```
:FILTER “/^MYDRIVER:/" red
```

Here we are using a regular expression. The ‘^’ at the beginning of the pattern specifies that the pattern can only match at the beginning of a line. The quotes that surround the regular expression are optional, but they would be required if the pattern contained spaces. We omitted the background color, so the normal text background (usually black) will be used. This filter will display all lines that begin with the text “MYDRIVER:” in red. Note that unless the `CaseSensitive` global is on, this will be a case-insensitive match.

A more interesting use of filters would be to set a breakpoint when a certain message is displayed. Assume you have a network driver called `nicMyDrv.sys` that handles network OIDs in a dispatch function called `QueryInformation`. Now imagine that for some reason you want to set a breakpoint on this function so that SoftICE will popup the first time it is called after an `OID_GEN_LINK_SPEED` is handled. Fortunately, your driver already displays the name of each OID as it is passed down, so you decide to use a filter. Here is the command (you would need to have symbols loaded for your driver in order for this to work) you would use:

```
:FILTER OID_GEN_LINK_SPEED red+white do  
"bpx nicMyDrv!QueryInformation;filter -d 0;filter -d 0;x;"
```

With this filter active, when your driver displays the name of `OID_GEN_LINK_SPEED`, the filter displays this line in red on white, and sets an execution breakpoint on the `QueryInformation` function in your driver (we specified the driver name before the symbol name just in case `nicMyDrv` was not the current symbol table). The second command in the action, `filter -d 0`, will delete filter 0, which will be this filter if no others have been defined. The `x;` command causes SoftICE to pop down after this command has been executed. So this filter will execute once, set a breakpoint, and delete itself, and the next time `QueryInformation` is called, SoftICE will pop up on the breakpoint we defined.

FKEY

Type

Customization

Definition

Show and edit the function key assignments.

Syntax

FKEY [*function-key string*]

function-key

Key	Description
<i>F1 - F12</i>	Unshifted function key.
<i>SF1 - SF12</i>	Shifted function key.
<i>CF1 - CF12</i>	Control key plus function key.
<i>AF1 - AF12</i>	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 SoftICE commands to a function-key. If you use the command without any parameters, the current function-key assignments display.

Note: 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 series of commands to a function-key. The carriage return symbol is represented by a semi-colon (;).

If you put a caret (^) or press Shift-6 in front of a command name, the command becomes invisible. You can use the command like any other, 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.

You can also use the plus sign (+) to assign an incomplete command to a function-key. When the function key is pressed, SoftICE displays the partial command in the command line so that the user can complete it.

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

Example

The following example assigns the command to toggle the Register window command (WR) 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, you can press the F2 key to toggle the Register window on or off.

```
FKEY f2 ^wr;
```

The following 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 cfl g cs:8028f000;d ss:esp;u cs:eip+
```

After you enter the following 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 assigns commands to the F1 key to toggle the Register window, create a Locals window of length 8, and a Code window of length 10.

```
FKEY f1 wr;wl 8;wc 10;
```

FLASH

Type

Window Control

Definition

Restore the Windows screen during *t* or *p* commands.

Syntax

```
FLASH [on | off]
```

Use

Use the FLASH command to specify whether the Windows screen restores during any *t* (trace) or *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 *t* or *p* command is executing. This feature is needed to debug sections of code that access video memory directly.

In most cases, if the routine being called writes to the Windows screen, and the *p* command executes across such a call, the screen restores. However, when you are debugging protected mode applications, with FLASH off, 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

The following command turns on FLASH mode. The Windows screen restores during any subsequent *t* or *p* commands.

```
FLASH on
```

See Also

SET

FMUTEX

Type

System Information

Definition

Display information about a mutant kernel object.

Syntax

FMUTEX [*expression*]

expression An expression that resolves to a valid address is acceptable.

Use

The FMUTEX command displays information about the mutant object identified by the expression you specify.

You must enter an expression to get data, because this is not itself a Windows NT- type object. The *expression* parameter is something that would not generally be considered a name. That is, it is a number, a complex expression (an expression which contains operators, such as Explorer + 0), or a register name.

Example

The following example displays information about the FMUTEX object:

FMUTEX ecx				
Address	Count	Own KTEB(TID)	Contention	OLDIql
State				
8014EA10	1	1(0P)	0	0
Clear				

See Also

KMUTEX

FOBJ

Type

System Information

Definition

Display information about a file object.

Syntax

FOBJ [*fobj-address*]

<i>fobj-address</i>	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 DDK. A few fields deserve special mention, however, because device driver writers find them particularly useful:

<i>DeviceObject</i>	This field is a pointer to the device object associated with the file object.
---------------------	---

<i>Vpb</i>	This is a pointer to the volume parameter block associated with the file object (if any).
------------	---

<i>FSContext1</i> and <i>FSContext2</i>	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 output from the FOBJ command.

```
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
```

FORMAT

Type

Window Control

Key

Shift-F3

Definition

Change the format of the active Data window.

Syntax

FORMAT

Use

Tip: The Data window format can also be changed by left-clicking the mouse with the cursor on the Data window's display format description (byte, word, dword, short real, long real, 10-byte real).

Use the FORMAT command to change the display format in the active Data window.

FORMAT cycles through the display formats in the following order: byte, word, dword, short real, long real, 10-byte real, and then byte again. Each call to FORMAT changes the window to the next display format in this order. This command is most useful when assigned to a function key.

The default function key assignment is Shift-F3. Shift-F3 is also supported while editing in the Data window.

Example

The following example changes the Data window to the next display format in the sequence: byte, word, dword, short real, long real, and 10-byte real.

FORMAT

G

Type

Flow Control

Definition

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.

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

Example

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

```
G 80123456
```

GDT

Type

System Information

Definition

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 a local descriptor table (LDT) selector (that is, bit 2 is a 1), SoftICE automatically displays information from the LDT, rather than the GDT.

Output

The base linear address and the 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 The lower two bits of this value reflects the descriptor privilege level.

selector type One of the following:

Type	Description
<i>Code16</i>	16-bit code selector.
<i>Data16</i>	16-bit data selector.
<i>Code32</i>	32-bit code selector.
<i>Data32</i>	32-bit data selector.
<i>LDT</i>	Local Descriptor Table selector.
<i>TSS32</i>	32-bit Task State Segment selector.
<i>TSS16</i>	16-bit Task State Segment selector.
<i>CallG32</i>	32-bit Call Gate selector.

Type	Description(Continued)
<i>CallG16</i>	16-bit Call Gate selector.
<i>TaskG32</i>	32-bit Task Gate selector.
<i>TaskG16</i>	16-bit Task Gate selector.
<i>TrapG32</i>	32-bit Trap Gate selector.
<i>TrapG16</i>	16-bit Trap Gate selector.
<i>IntG32</i>	32-bit Interrupt Gate selector.
<i>IntG16</i>	16-bit Interrupt Gate selector.
<i>Reserved</i>	Reserved selector.

<i>selector base</i>	Linear base address of the selector.
<i>selector limit</i>	Size of the selector's segment.
<i>selector DPL</i>	The selector's descriptor privilege level (DPL), which is either 0, 1, 2 or 3.
<i>present bit</i>	P or NP, indicating whether the selector is present or not present.
<i>segment attributes</i>	One of the following:

Value	Description
<i>RW</i>	Data selector is readable and writable.
<i>RO</i>	Data selector is read only.
<i>RE</i>	Code selector is readable and executable.
<i>EO</i>	Code selector is execute only.
<i>B</i>	TSS's busy bit is set.
<i>ED</i>	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
```

GENINT

Type

Flow Control

Definition

Force an interrupt to occur.

Syntax

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

nmi Forces a non-maskable interrupt.

int1 Forces an INT 1 interrupt.

int3 Forces an INT 3 interrupt.

interrupt-number 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, and 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.

Caution: You must make certain that there is a valid interrupt handler before using this command. SoftICE does not know if there is a handler installed. Your machine is likely to crash if you issue this command without a handler.

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

Example

When using CodeView for Windows, use the command:

```
GENINT 0
```


To pass control to 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. The following example shows how to avoid this situation. 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

Type

Miscellaneous

Key

F1

Definition

Display help information.

Syntax

H [*command*]

Use

The H command displays help on SoftICE commands. (Refer to the ? command on page 3 for information about evaluating expressions under the Windows NT family.) To display general help on all the SoftICE commands, enter the H command without 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

Type

Flow Control

Definition

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

The following command forces the system to reboot.

```
HBOOT
```

HEAP

Type

System Information

Definition

Display the Windows global heap.

Syntax

```
HEAP -L [FREE | mod | sel]
```

<i>-L</i>	Display only global heap entries that contain a local heap.
<i>FREE</i>	Display only heap entries marked as FREE.
<i>mod</i>	Name of the module.
<i>sel</i>	LDT selector.

Use

Note: For 16-bit modules, the HEAP command is process-specific. You must be in a NTVDM process that contains a WOW (Windows on Windows) box. Refer to *HEAP32* on page 119.

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 (Code Segment:Extension Index Pointer) 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.

For each heap entry the following information displays:

<i>selector or handle</i>	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.
<i>address</i>	32-bit flat virtual address.
<i>size</i>	Size of the heap entry in bytes.
<i>module name</i>	Module name of the owner of the heap entry.
<i>type</i>	Type of entry. One of the following:

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

The following example displays all heap entries belonging to the **KERNEL** module.

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

Refer to *HEAP32* on page 119.

HEAP32

Type

System Information

Definition

Display the Windows 32-bit heap.

Syntax

```
HEAP32 [[-w -x -s -v -b -trace] [heap32 | heap-entry | process-type]]
```

<i>-w</i>	Walk the heap, showing information about each heap entry.
<i>-x</i>	Show an extended summary of a 32-bit heap.
<i>-s</i>	Provide a segment summary for a heap.
<i>-v</i>	Validate a heap or heap-entry.
<i>-b</i>	Show base address and sizes of heap entry headers.
<i>-trace</i>	Display a heap trace buffer.
<i>heap32</i>	32-bit heap handle.
<i>heap-entry</i>	Heap allocated block returned by HeapAlloc or HeapRealloc.
<i>process-type</i>	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 116. This information only applies to 32-bit HEAP functionality.

Walk Option

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

Extended Option

The extended option (-x) 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.

Segment Option

The segment option (-s) displays a simple summary for the heap and for 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.

Validate Option

The validate option (-v) 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, SoftICE displays the following message:

```
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, SoftICE displays the following message:

```
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, SoftICE displays the following message:

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


Base Option

Use the base option (-b) 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. The size of the full heap-entry includes the heap-entry header, and any extra data allocated for guard-bytes or to satisfy alignment requirements. Under most circumstances you only specify base addressing when you need 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, because the heap-entry header precedes the actual heap-entry by 8 bytes. Secondly, the size for the allocated blocks is larger because it includes an additional 8 bytes for the heap-entry header, guard-bytes, and any extra bytes needed for proper alignment. The output from the base option is useful for manually navigating between adjacent heap entries, and for checking for memory overruns between the end of the heap-entry data and any unused space prior to the guard-bytes. The guard-bytes 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.

Trace Option

Use the trace option (-trace) 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 run 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.

The HEAP32 command and all of its options work on either a single specified heap handle or ALL the heaps for an entire process. For example, the following command performs a heap validation for all the heaps in the Test32 process:

```
HEAP 32 -v test32
```

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

In some cases, the actual physical memory that backs a particular linear address will not be present in memory, because it has been paged out by the operating system. In these cases, the HEAP32 command detects, avoids, and, where possible, continues to operate without the “not-present” pages. If 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. Since 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

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

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

<i>Base</i>	This is the address of the heap entry.
<i>Type</i>	Type of the heap entry.
<hr/>	
	Heap Entry Description
<hr/>	
<i>HEAP</i>	Represents the heap header.
<i>SEGMENT</i>	Represents a heap segment.
<i>ALLOC</i>	Active heap entry.
<i>FREE</i>	Inactive heap entry.
<i>VABLOCK</i>	Virtually-allocated block (VAB).
<hr/>	
<i>Size</i>	Size of the heap-entry. Typically, this is the number of bytes available to the application for data storage.
<i>Seg#</i>	Heap segment in which the heap-entry is allocated.
<i>Flags</i>	Heap entry flags.

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

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

Example

The following example displays a basic heap summary for every heap in every process.

HEAP32					
Base	Id	Cmmt/Psnt/Rsvd	Segments	Flags	Process
00160000	01	0006/0006/00FA	1	00000002	smss
00160000	02	0012/0011/00EE	1	00000002	csrss
7F6F0000	03	0007/0007/00F9	2	00005002	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 general information on the HEAP command, refer to *HEAP* on page 116.

HERE

Type

Flow Control

Key

F7

Definition

Go to the current cursor line.

Syntax

HERE

Use

When the cursor is in the Code window, 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 to which you want to execute. Enter HERE or press the programmed function key for HERE (F7 by default).

The HERE command sets a single, one-time execution breakpoint set at the address of the current cursor position, arms all sticky breakpoints, and exits from SoftICE.

Execution begins at the current CS:EIP (Code Segment:Execution Index Pointer) and continues until the execution breakpoint is encountered, the window pop-up key sequence is used, or a sticky breakpoint occurs. When SoftICE pops up, for any of these reasons, 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.

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

Example

The following command sets an execution breakpoint at the current cursor position, exits from SoftICE, and begins execution at the current CS:EIP.

```
HERE
```

HS

Type

System Information

Definition

Search the history buffer for the specified string.

Syntax

HS [- | +] string

- (*minus sign*) Search backwards from this point.

+ (*plus sign*) Search forwards from this point.

string Specified search string.

Use

You can search forward (which is the default) using the '+', or backward, using '-'. If you enter this command without parameters, SoftICE uses the previous search, starting from the last string found.

Use single quotation marks to search for text that includes spaces.

Example

Enter the following command to find the first load notifications for the net module in the history buffer.

```
HS 'load32 mod=net'
```

See Also

S, SS

HWND

Type

System Information

Definition

Display information on Window handles.

Syntax

<code>HWND [-x][-c] [<i>hwnd-type</i> <i>desktop-type</i> <i>process-type</i> <i>thread-type</i> <i>module-type</i> <i>class-name</i>]</code>	
<code>-x</code>	Extended. Display extended information about each window handle.
<code>-c</code>	Children. Force the display of the window hierarchy when searching by thread-type, module-type, or class-name.
<i>hwnd-type</i>	Window handle or pointer to a window structure.
<i>desktop-type</i>	Desktop handle or desktop pointer to a window structure (3.51 only).
<i>process-type, thread-type, or module-type</i>	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.
<i>class name</i>	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.

The extended option (-x) shows extended information about each window.

When you specify the extended option, or an owner-type (process-type, thread-type, or module-type) as a parameter, the HWND command will not automatically enumerate child windows. Specifying the children option (-c) 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:

<i>Handle</i>	HWND handle (refer to <i>OBJTAB</i> on page 191 for more information). Each window handle is indented to show its child and sibling relationships to other windows.
<i>Class</i>	Registered class name for the window, if available (refer to <i>CLASS</i> on page 45 for more information).
<i>WinProc</i>	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.
<i>TID</i>	Owning thread ID.
<i>Module</i>	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      ClassWinProcTID  Module
01001E      #32769 (Desktop)5FBFE42524winsrv
050060      #32770 (Dialog)60A2930418winlogon
010044      SAS window class022A49C418winlogon
010020      #32768 (PopupMenu)5FBEDBD524winsrv
010022      #32769 (Desktop)5FBFE42524winsrv
010024      #32768 (PopupMenu)5FBEDBD524winsrv
030074      Shell_TrayWnd0101775E67Explorer
030072      Button01012A4E67Explorer
0800AA      TrayNotifyWnd010216C467Explorer
03003E      TrayClockWClass01028C8567Explorer
030078      MSTaskSwWClass01022F6967Explorer
030076      SysTabControl32712188A867Explorer
05007A      tooltips_class327120B43A67Explorer
03003C      tooltips_class327120B43A67Explorer
2E00F0      NDDEAgnt016E18F14Bnddeagnt
1C0148      CLIPBOARDWNDCLASS034F:29182COLE2
9B0152      DdeCommonWindowClass77C2D88B2Cole32
3200F2      OleObjectRpcWindow77C2D73B2Cole32
0800A2      DdeCommonWindowClass77C2D88B67ole32
030086      OleMainThreadWndClass77C2DCF267ole32
030088      OleObjectRpcWindow77C2D73B67ole32
03008A      ProxyTarget71E6869A67shell32
03008C      ProxyTarget71E6869A67shell32
030070      ProxyTarget71E6869A67shell32
04007C      ProxyTarget71E6869A67shell32
0400CC      OTCClass0100D7F367Explorer
0300CA      DDEMLEvent5FC216AB67winsrv
0300C6      DDEMLMom60A2779D6700000000
0300C0      #420BB7:077678MMSYSTEM
0300D2      WOWFaxClass01F9F7A878WOWEXEC
060062      ConsoleWindowClass5FCD23C72Bwinsrv
0300B4      WOWExecClass03CF:0B3E78WOWEXEC
030068      Progman0101B1D367Explorer
0E00BC      SHELLDLL_DefView71E300E867shell32
040082      SysListView327121A0EC67shell32
030080      SysHeader327120B06F67shell32
```

Notes: The output from the previous example enumerates 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.

Since the system may 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 with a specific window handle.

```
HWND 400a0
Handle      ClassWinProcTIDModule
0400A0      Progman0101B1D374Explorer
```

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

```
HWND 74
Handle      ClassWinProcTIDModule
2F00F0      Shell_TrayWnd0101775E74Explorer
0500CE      Button01012A4E74Explorer
0500C4      TrayNotifyWnd010216C474Explorer
040074      TrayClockWClass01028C8574Explorer
0500C6      MSTaskSwWClass01022F6974Explorer
0400C8      SysTabControl32712188A874Explorer
3700F2      tooltips_class327120B43A74Explorer
040066      tooltips_class327120B43A74Explorer
0F00BC      DdeCommonWindowClass77C2D88B74ole32
040068      OleMainThreadWndClass77C2DCF274ole32
0500CC      OleObjectRpcWindow77C2D73B74ole32
2600BA      ProxyTarget71E6869A74shell32
0400D0      ProxyTarget71E6869A74shell32
0400CA      ProxyTarget71E6869A74shell32
070094      ProxyTarget71E6869A74shell32
04009E      OTClass0100D7F374Explorer
480092      DDEMLEvent5FC216AB74winsrv
09004A      DDEMLMom60A2779D7400000000
0400A0      Progman0101B1D374Explorer
0500C0      SHELLDLL_DefView71E300E874shell32
070090      SysListView327121A0EC74shell32
050096      SysHeader327120B06F74shell32
```

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 (-x) 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)
```

Type

I/O Port

Definition

Input a value from an I/O port.

Syntax

`I[size] port`

size

Value	Description
<i>B</i>	Byte
<i>W</i>	Word
<i>D</i>	DWORD

port

Port Address.

Use

You use the I command to read and display a value from a specified hardware port. Input can be done from byte, word, or dword ports. If you do not specify size, the default is byte.

Except for the interrupt mask registers, the I command does an actual I/O instruction, so it displays the actual state of the hardware port. However, in the case of virtualized ports, the actual data returned by the I command may not be the same as the virtualized data that an application would see.

The only ports that SoftICE does not perform 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.

Example

The following example performs an input from port 21, which is the mask register for interrupt controller one.

I 21

I1HERE

Type

Mode Control

Definition

Pop up on embedded INT 1 instructions.

Syntax

```
I1HERE [on | off]
```

Use

Use the I1HERE command to specify that any embedded interrupt 1 instruction brings up the SoftICE screen. This feature is useful for stopping your program in a specific location. When I1HERE is on, SoftICE checks to see whether an interrupt is really an INT 1 in the code before popping up. If it is not an INT 1, 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 3s for breakpoint notifications, I1HERE allows you to use INT 1s as hardwired interrupts in your code without triggering the application debugger.

Example

The following example turns on I1HERE mode. Any INT 1s generated after this point bring up the SoftICE screen.

```
I1HERE on
```

I3HERE

Type

Mode Control

Definition

Pop up on INT 3 instructions.

Syntax

I3HERE [**on** | **off** | **DRV**]

DRV Enable INT 3 handling above 2GB only. This supports trapping of a driver's call to DebugBreak().

Use

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

To use this feature, set I3HERE on, and place an INT 3 instruction 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.

When 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 Compuware's BoundsChecker, you should place INT 1 instructions in your code instead of INT 3 instructions. See *I1HERE* on page 133.

Example

The following 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. The following example shows how to avoid this situation. 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; I1HERE; SET

IDT

Type

System Information

Definition

Display the Interrupt Descriptor Table.

Syntax

IDT [*int-number* | *IDT base-address*]

int-number Interrupt number to display information.

IDT base-address IDT's base address and limit.

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 base address and limit of the IDT, as well as the contents of all entries in the table. If you specify an optional interrupt-number or an IDT base address, only information about that entry is displayed.

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 to view symbol names. (See *TABLE* on page 247.)

Output

Each line of the display contains the following information:

interrupt number 0 - 0FFh.

interrupt type One of the following:

Type	Description
<i>CallG32</i>	32-bit Call Gate
<i>CallG16</i>	16-bit Call Gate
<i>TaskG</i>	Task Gate

Type	Description(Continued)
<i>TrapG16</i>	16-bit Trap Gate
<i>TrapG32</i>	32-bit Trap Gate
<i>IntG32</i>	32-bit Interrupt Gate
<i>IntG16</i>	16-bit Interrupt Gate

<i>address</i>	Selector:offset of the interrupt handler.
<i>selector's DPL</i>	Selector's descriptor privilege level (DPL), which is either 0, 1, 2 or 3.
<i>present bit</i>	P or NP, indicating whether the entry is present or not present.
<i>Owner+Offset</i>	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 following 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	

INTOBJ

Type

System Information

Definition

Display information on system interrupt objects.

Syntax

```
INTOBJ [ vector | interrupt-object-address ]
```

Use

The INTOBJ command displays information about interrupt objects that are current in the system. If you enter INTOBJ without parameters, SoftICE lists all interrupt objects with the following information:

- ◆ Object Address
- ◆ Vector
- ◆ Service Address
- ◆ Service Context
- ◆ IRQL
- ◆ Mode
- ◆ Affinity Mask
- ◆ Symbol

If you issue the command with a vector or address, SoftICE displays information about the specified interrupt object.

Example

The following example displays information about all the current interrupt objects in the system.

INTOBJ						
Object		Service	Service			Affinity
Address	Vector	Address	Context	IRQL	Mode	Mask
Symbol						
807D0D88	31	80802D90	807D1030	1A	Edge	01
80750D88	33	808030F0	807500F8	18	Edge	01
80750B08	34	808030F0	807513F8	17	Edge	01
807E0968	35	80802D30	807E1008	16	Edge	01
807E28A8	39	80802D50	807E9C48	12	Edge	01
80792D88	3B	80802ED0	8078D158	10	Level	01
807D18C8	3C	80802D70	807D1030	0F	Edge	01
808F2428	3E	8022BF58	808F2850	0D	Edge	01
SCSI!PORT!.text+0C98						
807EB428	3F	8022BF58	807EB850	0C	Edge	01
SCSI!PORT!.text+0C98						

The following example shows the information SoftICE displays for a particular interrupt object:

INTOBJ 31
Interrupt Object at 807D0D88
Length: 01E4
List Forward Link: 807D0D8C
Object List Back Link: 807D0D8C
Interrupt Service Routine address: 80802D90
Interrupt Service Routine context: 807D1030
Spinlock: 807D155C
Vector: 31
Device IRQL: 1A
Save Floating Point: FALSE
Processor Affinity Mask: 01
Processor Number: 00
Share interrupt: TRUE
Interrupt mode: Edge

IOCTL

Type

System Information

Definition

Decodes the IOCTL and FSCTL codes into readable English.

Syntax

IOCTL *iocode*

iocode Numeric value, register, or symbol.

Use

This command decodes an IOCTL code or an FSCTL code its corresponding Device Type, Function #, Method, and Access type.

Note: This routine will only decode Microsoft publicly-defined codes (as defined in the DDK's WinIoCtl.h). It does not decode user-defined IOCTL codes.

Example

The following example shows the output from the IOCTL command.

```
:IOCTL 22200
Control Code: 0x00022200
Device Type:  FILE_DEVICE_CD_ROM
Function:      0x880 (2176) (User Function)
Method:        METHOD_BUFFERED
Access:        FILE_ANY_ACCESS
```

IRB

Type

System Information

Definition

Decodes a 1394 I/O Request Block (IRB).

Syntax

```
IRB [-r] pirb
```

-r Displays the reserved fields.

pirb Pointer to the IRB.

Use

The IRB command will interpret and parse a 1394 IRB packet. The information that it displays is based on the information contained in the DDK header file 1394.h.

Note: The address specified will be assumed to be an IRB packet. There is no validation done, and it is the responsibility of the user to locate the IRB packet address.

Example

Some examples of IRB command usage are shown on the following two pages.

```

:IRB 0xFFA40C68
IRB at address FFA40C68
  FunctionNumber=0x6  (REQUEST_ISOCH_ATTACH_BUFFERS)
  Flags=0x0

  hResource           : 80CDB008
  nNumberOfDescriptors : 00000001
  pIsochDescriptor
    ISOCH_DESCRIPTOR (1 of 1)
      fulFlags         : 00000001
      Md1              : 80D48578
      ulLength         : 00025800
      nMaxBytesPerFrame : 00000280
      ulSynch          : 00000001
      ulTag            : 00000000
      CycleTime
        CYCLE_TIME
          CL_CycleCount      : 00000000
          CL_CycleOffset    : 00000000
          CL_SecondCount    : 00000000
      Callback         : FC602C50
      Context1         : 80DAA390
      Context2         : FFA40C68
      status           : 00000000
      DeviceReserved   [0] : 00000001
      DeviceReserved   [1] : 80D422B0
      DeviceReserved   [2] : FF9F08D8
      DeviceReserved   [3] : 00000000
      DeviceReserved   [4] : 00000000
      DeviceReserved   [5] : 00000000
      DeviceReserved   [6] : 00000000
      DeviceReserved   [7] : 00000000

```

```

:IRB -r 0xFFA40C68
IRB at address FFA40C68
  FunctionNumber=0x6  (REQUEST_ISOCH_ATTACH_BUFFERS)
  Flags=0x0

  hResource           : 80CDB008
  nNumberOfDescriptors : 00000001
  pIsochDescriptor
    ISOCH_DESCRIPTOR (1 of 1)
      fulFlags         : 00000001
      Mdl              : 80D48578
      ulLength         : 00025800
      nMaxBytesPerFrame : 00000280
      ulSynch          : 00000001
      ulTag            : 00000000
      CycleTime
        CYCLE_TIME
          CL_CycleCount      : 00000000
          CL_CycleOffset    : 00000000
          CL_SecondCount     : 00000000
      Callback          : FC602C50
      Context1          : 80DAA390
      Context2          : FFA40C68
      status            : 00000000
      DeviceReserved    [0] : 00000001
      DeviceReserved    [1] : 80D422B0
      DeviceReserved    [2] : FF9F08D8
      DeviceReserved    [3] : 00000000
      DeviceReserved    [4] : 00000000
      DeviceReserved    [5] : 00000000
      DeviceReserved    [6] : 00000000
      DeviceReserved    [7] : 00000000
      BusReserved       [0] : 00000000
      BusReserved       [1] : 00000000
      BusReserved       [2] : 00000000
      BusReserved       [3] : 00000000
      BusReserved       [4] : 00000000
      BusReserved       [5] : 00000000
      BusReserved       [6] : 00000000
      BusReserved       [7] : 00000000
      PortReserved      [0] : 00000000
      PortReserved      [1] : 00000000
      PortReserved      [2] : 00000000
      PortReserved      [3] : 00000000
      PortReserved      [4] : 00000000
      PortReserved      [5] : 00000000
      PortReserved      [6] : 00000000
      PortReserved      [7] : 00000000
:

```

See Also

IRP

IRP

Type

System Information

Definition

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

Syntax

IRP [**-f** | **-n** | **-p** | **-a**] *pirp*

<i>-f</i>	Display all IRP stack locations.
<i>-n</i>	Display the next IRP stack location.
<i>-p</i>	Walk the previous IRP stack location.
<i>-a</i>	Iterates through all threads on a system and shows the IRP for each thread.
<i>pirp</i>	Pointer to 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. Note that the command does not check the validity of the IRP structure at the specified address, so any address will be accepted by SoftICE as an IRP address. Be careful to pass the IRP command a valid IRP address.

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

<i>Flags</i>	Flags used to define IRP attributes.
<i>StackCount</i>	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.
<i>CurrentLocation</i>	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.

<i>Cancel</i>	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.
<i>Tail.Overlay. CurrentStackLoc</i>	Address of current stack location. The contents of this stack location are displayed after the IRP, as illustrated in the example of the command given below.
<i>Cancel</i>	This boolean is set to TRUE if the IRP has been cancelled as a result of an IRP cancellation call. An IRP may be cancelled when the IRP's result is no longer needed so that the IRP will not complete.

These fields in the current stack location may be useful:

<i>Major Function and Minor Function</i>	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.
<i>Device Object</i>	Pointer to the device object at which the IRP is currently stationed. 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.
<i>File Object</i>	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.
<i>Completion Routine</i>	This field is set when a driver sets a completion routine for an IRP through the <i>IoSetCompletionRoutine</i> 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 <i>IoCompleteRequest</i> .

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
```

See Also

IRB

IRQ

Type

System Information

Definition

Display information about system hardware interrupts (IRQs).

Syntax

IRQ [*irq-number*]

irq-number Specific IRQ to be displayed.

Use

The IRQ command will display information about the hardware interrupts (IRQs) in the system. Issuing the IRQ command with no parameters will display a list of all the hardware interrupts on the system, along with assigned vector and status information.

The output from this command differs depending on whether the machine is equipped with an 8259-style PIC, or an APIC. On a PIC machine, the IRQ command will report the vector and status (masked/unmasked) for each IRQ in the system. The vector field is an index into the system's Interrupt Descriptor Table, and can be used with the SoftICE IDT command to locate the interrupt service routine associated with a hardware interrupt. The status field indicates whether the hardware interrupt is currently masked at the interrupt controller.

On APIC systems, the IRQ command gets its information by reading the I/O APIC. The command displays the vector, delivery mode, status, trigger mode, and destination information for each IRQ. As with the PIC version of the IRQ command, the vector number is an offset into the system's IDT, and can be used to locate the interrupt service routine for the hardware interrupt. For a complete explanation of the other information reported by the IRQ command, refer to the I/O APIC documentation.

Example

The following example shows the output from the IRQ command on a machine equipped with a PIC:

IRQ		
IRQ	Vector	Status
00	30	Unmasked
01	31	Unmasked
02	32	Unmasked
03	33	nmasked
04	34	Unmasked
05	35	Masked
06	36	Masked
07	37	Masked
08	38	Unmasked
09	39	Unmasked
0A	3A	Masked
0B	3B	Masked
0C	3C	Unmasked
0D	3D	Masked
0E	3E	Unmasked
0F	3F	Unmasked

And here is the output from the IRQ command on an APIC machine:

IRQ						
Inti	Vector	Delivery	Status	Trigger	Dest	Mode
Destination						
01	93	Low. Pri	Idle	Edge	Logical	0 1
03	B2	Low. Pri	Idle	Edge	Logical	0 1
04	92	Low. Pri	Idle	Edge	Logical	0 1
08	D1	Fixed	Idle	Edge	Logical	0
09	B1	Low. Pri	Pending	Level	Logical	0 1
0E	62	Low. Pri	Idle	Edge	Logical	0 1
0F	82	Low. Pri	Idle	Edge	Logical	0 1
10	83	Low. Pri	Pending	Level	Logical	0 1
11	63	Low. Pri	Idle	Level	Logical	0 1
13	B4	Low. Pri	Pending	Level	Logical	0 1
17	73	Low. Pri	Pending	Level	Logical	0 1
I/O unit id register: 02000000						
I/O unit version register: 00178020						

One use of the IRQ command is in identifying the interrupt service routine associated with a particular device in the system. This final example illustrates using SoftICE commands to determine the address of the ISR for a USB host controller.

First, the USB command returns the PCI addresses of all the USB host controllers in the system:

```
USB
3 USB Host Controllers Found
  HC 0: UHCI at PCI Bus 0 Device 1F Function 2
  HC 1: UHCI at PCI Bus 0 Device 1F Function 4
  HC 2: OHCI at PCI Bus 4 Device F Function 0
```

Next, we use the PCI command to determine the interrupt line assigned to the host controller by the OS. In this case, we'll get the interrupt line for the first of the three host controllers listed above:

```
PCI 0 1f 2
Bus 00 Device 1F Function 02
  Vendor: 8086Intel Corporation
  Device: 2442
  Revision: 04
  Device class: 0C Serial bus controller
  Device subclass: 03 Universal Serial Bus controller
  Device sub-subclass: 00
  Base address 4: 0000FF80 32 bytes I/O
  Interrupt line: 13 Interrupt pin: 04 Min_Gnt: 00
  MaxLat: 00
  Cache line size: 00 Latency timer: 00 Header type: 00
  BIST: 00
  Command Register:
    I/O:1 Mem:0 BusMast:1 Special:0 MemInv:0
    Parity:0 Wait:0 SERR:0 Back2Back:0 Snoop:0
  Status Register:
    Caps:0 66MHz Cap:0 UDF:0 FB2B Cap:1 DevSel: Medium
    PERRDet:0 PERRRcvd:0 TASgnld:0 TARcvd:0 MARcvd:0
    SERRSgnld:0
```

Notice that the interrupt line (**bolded in this example**) is set to 13 for this device. Now we issue the IRQ command, specifying the interrupt line from the PCI command:

```
IRQ 13
Inti Vector Delivery Status Trigger Dest Mode
Destination
13 B4 Low. Pri Pending Level Logical 0 1
I/O unit id register: 02000000
I/O unit version register: 00178020
```

This tells us that the interrupt vector assigned to this device is B4. Finally, we use the IDT command to get the address of the ISR:

```
IDT b4
```

Int	Type	Sel:Offset	Attributes	Symbol/Owner
00B4	IntG32	0008:827D8BEC	DPL=0 P	

The IDT command shows that the ISR address for this USB host controller is **0008:827D8BEC**.

KEVENT

Type

System Information

Definition

Display Kernel Events.

Syntax

KEVENT [*kernel-event*]

kernel-event Kernel event address.

Use

The KEVENT command displays information about kernel events that are current in the system. If you enter KEVENT without parameters, SoftICE walks through the BaseNamedObjects directory, where the Win32 subsystem typically stores named kernel objects, and displays the Kernel Events in that list. If you specify a kernel event address, SoftICE displays information about the specified event.

Example

The following example shows how to use the KEVENT command to display information about a specific event.

```
KEVENT 807AB730
Address  Type           State      Name
807AB730 Notification   Signalled  LSA_RPC_SERVER_ACTIVE
```

See Also

KMUTEX; KSEM

KMUTEX

Type

System Information

Definition

Display information about kernel mutexes.

Syntax

KMUTEX [*kernel-mutex*]

kernel-mutex Kernel mutex address.

Use

If you issue the KMUTEX command without any parameters, SoftICE walks through the BaseNamedObjects directory, where the Win32 subsystem typically stores named kernel objects, and displays information about all the Kernel mutexes in that list.

If you issue the KMUTEX command with an expression, SoftICE displays information about the kernel mutex at that address.

Example

The following example shows how to use the KEVENT command to display information about a specific object.

```
KMUTEX 80733470
Address  State      Own.KTEB(TID) Aban APC Name
80733470 Signalled      0( 0) N      0  OLESharedTablesMutex
```

See Also

FMUTEX; KEVENT; KSEM

KSEM

Type

System Information

Definition

Display information about kernel semaphores.

Syntax

KSEM [*semaphore-address*]

semaphore

-address Address of a kernel semaphore object.

Use

If you issue the KSEM command without any parameters, SoftICE walks through the BaseNamedObjects directory, where the Win32 subsystem typically stores named kernel objects, and displays information about all the Kernel semaphores in that list.

If you issue the KSEM command with an expression, SoftICE displays information about the kernel semaphores at that address.

Example

The following example shows how to use the KSEM command to display information about a specific semaphore object.

```
KSEM 807060F0
Address  Limit      State      Name
807060F0 1          Signalled NDDEAgent
```

See Also

KEVENT; KMUTEX

LDT

Type

System Information

Definition

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 Global Descriptor Table (GDT) selector (that is, 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.

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.

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:

<i>selector value</i>	Lower two bits of this value reflect the descriptor privilege level.																												
<i>selector type</i>	<table> <tr> <th>Type</th><th>Description</th></tr> <tr> <td><i>Code16</i></td><td>16-bit code selector</td></tr> <tr> <td><i>Data16</i></td><td>16-bit data selector</td></tr> <tr> <td><i>Code32</i></td><td>32-bit code selector</td></tr> <tr> <td><i>Data32</i></td><td>32-bit data selector</td></tr> <tr> <td><i>CallG32</i></td><td>32-bit Call Gate selector</td></tr> <tr> <td><i>CallG16</i></td><td>16-bit Call Gate selector</td></tr> <tr> <td><i>TaskG32</i></td><td>32-bit Task Gate selector</td></tr> <tr> <td><i>TaskG16</i></td><td>16-bit Task Gate selector</td></tr> <tr> <td><i>TrapG32</i></td><td>32-bit Trap Gate selector</td></tr> <tr> <td><i>TrapG16</i></td><td>16-bit Trap Gate selector</td></tr> <tr> <td><i>IntG32</i></td><td>32-bit Interrupt Gate selector</td></tr> <tr> <td><i>IntG16</i></td><td>16-bit Interrupt Gate selector</td></tr> <tr> <td><i>Reserved</i></td><td>Reserved selector</td></tr> </table>	Type	Description	<i>Code16</i>	16-bit code selector	<i>Data16</i>	16-bit data selector	<i>Code32</i>	32-bit code selector	<i>Data32</i>	32-bit data selector	<i>CallG32</i>	32-bit Call Gate selector	<i>CallG16</i>	16-bit Call Gate selector	<i>TaskG32</i>	32-bit Task Gate selector	<i>TaskG16</i>	16-bit Task Gate selector	<i>TrapG32</i>	32-bit Trap Gate selector	<i>TrapG16</i>	16-bit Trap Gate selector	<i>IntG32</i>	32-bit Interrupt Gate selector	<i>IntG16</i>	16-bit Interrupt Gate selector	<i>Reserved</i>	Reserved selector
Type	Description																												
<i>Code16</i>	16-bit code selector																												
<i>Data16</i>	16-bit data selector																												
<i>Code32</i>	32-bit code selector																												
<i>Data32</i>	32-bit data selector																												
<i>CallG32</i>	32-bit Call Gate selector																												
<i>CallG16</i>	16-bit Call Gate selector																												
<i>TaskG32</i>	32-bit Task Gate selector																												
<i>TaskG16</i>	16-bit Task Gate selector																												
<i>TrapG32</i>	32-bit Trap Gate selector																												
<i>TrapG16</i>	16-bit Trap Gate selector																												
<i>IntG32</i>	32-bit Interrupt Gate selector																												
<i>IntG16</i>	16-bit Interrupt Gate selector																												
<i>Reserved</i>	Reserved selector																												
<i>selector base</i>	Linear base address of the selector.																												
<i>selector limit</i>	Size of the selector.																												
<i>selector DPL</i>	Selector's descriptor privilege level (DPL), either 0, 1, 2 or 3.																												
<i>present bit</i>	P or NP, indicating whether the selector is present or not present.																												
<i>segment attributes</i>	One of the following: <table> <tr> <th>Type</th><th>Description</th></tr> <tr> <td><i>RW</i></td><td>Data selector is readable and writable.</td></tr> <tr> <td><i>RO</i></td><td>Data selector is read only.</td></tr> <tr> <td><i>RE</i></td><td>Code selector is readable and executable.</td></tr> <tr> <td><i>EO</i></td><td>Code selector is execute only.</td></tr> <tr> <td><i>B</i></td><td>TSS's busy bit is set.</td></tr> </table>	Type	Description	<i>RW</i>	Data selector is readable and writable.	<i>RO</i>	Data selector is read only.	<i>RE</i>	Code selector is readable and executable.	<i>EO</i>	Code selector is execute only.	<i>B</i>	TSS's busy bit is set.																
Type	Description																												
<i>RW</i>	Data selector is readable and writable.																												
<i>RO</i>	Data selector is read only.																												
<i>RE</i>	Code selector is readable and executable.																												
<i>EO</i>	Code selector is execute only.																												
<i>B</i>	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

Type

System Information

Definition

Display the Windows local heap.

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

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

Output

For each local heap entry the following information displays:

<i>offset</i>	16-bit offset relative to the specified selector base address.
<i>size</i>	Size of the heap entry in bytes.
<i>type</i>	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

The following command displays all local heap entries belonging to the GDI default local heap.

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

Type

Customization

Definition

Change the number of lines for the SoftICE display.

Syntax

With Universal Video Driver:

LINES numlines

numlines Number of screen lines. Set this to any value from 25 to 128.

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 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. You can display 60 lines for single monitor debugging. When debugging in serial mode, all line counts are supported for VGA displays.

Example

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

```
LINES 53
```

See Also

SET; WIDTH

LOCALS

Type

Symbol/Source Command

Definition

List 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 the local variable 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.

Note: 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. 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

Type

Miscellaneous

Definition

Move data.

Syntax

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

source-address Start of address range to move.

l 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

The following command moves 2000h bytes (8KB) from memory location DS:1000h to ES:5000h.

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

MACRO

Type

Customization

Definition

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

Syntax

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

<i>macro-name</i>	Case-insensitive, 3-8 character name for the macro being defined, or the name of an existing macro.
<i>macro-body</i>	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 three and eight characters long, and may contain any alphanumeric character and underscores (_). If the macro-name parameter specifies an existing macro, the MACRO command redefines the existing macro. 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 (\). Since 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 its 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.

Two special macros can be defined using this command: POPUP and POPDOWN. These macros, if defined, are automatically executed every time SoftICE pops up or down.

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 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 example uses 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"  
lshot    = "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 examples show other basic uses for the MACRO command:

Command	Description
MACRO *	Delete all named macros.
MACRO oops *	Delete the macro named <i>oops</i> .
MACRO xwhat	Edit the macro named <i>xwhat</i> .

Note: Since 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 example is a simple 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 the previous 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 the revised 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 lshot = "bpx %1 do \"bc bindex\""" lshot eip
or
lshot @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"~~

Corrected Example: MACRO dda = "dd dataaddr"

Explanation: The macro name is a duplication of a SoftICE command name. SoftICE commands cannot be redefined.

Illegal Definition: ~~MACRO aa = "addr %1"~~

Illegal Definition:	MACRO dd = "dd dataaddr"
Corrected Example:	MACRO aaa = "addr %1"
Explanation:	The macro command name is too short. A macro name must be between 3 and 8 characters long.
Illegal Definition:	MACRO pbsz = ? hibyte(hiword(*(%1-8))) << 5
Corrected Example:	MACRO pbsz = "? hibyte(hiword(*(%1-8))) << 5"
Explanation:	The macro body must be surrounded by quote characters (").
Illegal Definition:	MACRO tag = "? *(%2-4)"
Corrected Example:	MACRO tag = "? *(%1-4)"
Explanation:	The macro body references parameter %2 without referencing parameter %1. You cannot reference parameter %n+1 without having referenced parameter %n.

The following shows how to setup the POPUP macro, which is automatically executed whenever SoftICE pops up, to display memory pointed to by the esi register.

```
MACRO popup="dd esi"
```

The POPDOWN macro can be defined in the same way, and executes whenever SoftICE is returning control to the OS.

MAP32

Type

System Information

Definition

Display a memory map of all 32-bit modules currently loaded in memory.

Syntax

```
MAP32 [-u | -s][module-handle | module-name | address]
```

<i>-u</i>	Displays only modules in user space.
<i>-s</i>	Displays only modules in system space.
<i>module-handle</i>	Base address of a module image.
<i>module-name</i>	Windows module-name.
<i>address</i>	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 specified 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
```

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 the output of MAP32 contains the following information:

<i>Owner</i>	Module name.
<i>Name</i>	Section name from the executable file.

<i>Obj#</i>	Section number from the executable file.
<i>Address</i>	Selector:offset address of the section.
<i>Size</i>	Section's size in bytes.
<i>Type</i>	Type and attributes of the section, as follows:

Type	Attributes
<i>CODE</i>	Code.
<i>IDATA</i>	Initialized Data.
<i>UDATA</i>	Uninitialized Data.
<i>RO</i>	Read Only.
<i>RW</i>	Read/Write.
<i>SHARED</i>	Object is shared.

Example

The following example illustrates sample output for MAP32 executed on a Win2K platform.

MAP32					
Owner	Obj Name	Obj#	Address	Size	Type
ntoskrnl	.text	0001	0008:00400500	0006A1BF	CODE RO
ntoskrnl	POOLCODE	0002	0008:0046A6C0	00000B35	CODE RO
ntoskrnl	POOLMI	0003	0008:0046B200	0000129B	CODE RO
ntoskrnl	MISYSPTE	0004	0008:0046C4C0	000006EC	IDATA RO
ntoskrnl	.data	0005	0023:0046CBC0	00017B64	CODE RO
ntoskrnl	PAGELK	0006	0008:00484740	0000B8B3	CODE RO
ntoskrnl	PAGE	0007	0008:00490000	0009F591	CODE RO

MAPV86

Type

System Information

Definition

Display the memory map of the current Virtual Machine (NTVDM).

Note: The MAPV86 command is process specific. You must be in an NTVDM process, because this process is the only one that can contain V86 boxes. There is no *global* MS-DOS used in the Windows NT family.

Syntax

MAPV86 [*address*]

address Segment:offset type address.

Use

If no address parameter is specified, a map of the entire current NTVDM'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 NTVDM 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. You can enable source-level debugging for these DOS programs by performing the following steps.

- ◆ Use the MAPV86 command to get the starting address of the programs static code segment. Add 10h to the address if the program in an executable (.EXE).
- ◆ Use the SYMLOC command to set the symbol table alignment to that value.

Output

The following summary information is displayed by the MAPV86 command in a NTVDM process.

<i>VM ID</i>	Virtual machine (VM) ID. VM1 is the System VM.
<i>VM handle</i>	32-bit virtual machine handle.
<i>CRS pointer</i>	VM's 32-bit client register structure pointer.
<i>VM address</i>	32-bit linear address of the VM. This is the <i>high linear</i> address of the virtual machine, which is also currently mapped to linear address 0.

If the current CS:EIP belongs to a MAPV86 entry, that line will be highlighted. Each line of the MAPV86 display contains the following information:

<i>Start</i>	Segment:offset start address of the component.
<i>Length</i>	Length of the component in paragraphs.
<i>Name</i>	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 NTVDM.

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

Type

System Information

Definition

Display the Windows module list.

Syntax

```
MOD [-u | -s] | [partial-name* | address]
```

<i>-u</i>	Displays only modules in user space.
<i>-s</i>	Displays only modules in system space.
<i>partial-name*</i>	Prefix of the Windows module name.
<i>address</i>	Any address that falls within an executable image.

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:

- ◆ 16-bit modules
- ◆ 32-bit driver modules
- ◆ 32-bit application modules

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:

<i>module handle</i>	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 is only displayed in this column for 32-bit modules.

pe-header Selector:offset of the PE File header for that module.
Note: A value is only 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 abbreviated example shows MOD on the NTVDM WOW process:

MOD			
hMod	Base	PEHeader	Module Name 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	ntoskrnl	\WINNT35\System32\ntoskrnl.exe
	80100080		
	80400000	hal	\WINNT35\System32\hal.dll
	80400080		
	80010000	atapi	atapi.sys
	80010080		

80013000	SCSIPOINT	\WINNT35\System32\Drivers\SCSIPO
80013080		RT.SYS
80001000	Atdisk	Atdisk.sys
80001080		
8001B000	Scsidisk	Scsidisk.sys
8001B080		
803AE000	Fastfat	Fastfat.sys
803AE080		
FB000000	Floppy	\SystemRoot\System32\Drivers\Flo
FB000080		ppy.SYS
FB010000	Scsicdrm	\SystemRoot\System32\Drivers\Scs
FB010080		icdrm.SYS
FB020000	Fs_Rec	\SystemRoot\System32\Drivers\Fs_
FB020080		Rec.SYS

MSR

Type

System Information

Definition

Display or write to the Model Specific Registers.

Syntax

```
MSR [[-u] [begin-reg [end-reg] | [-w reg [hidword_val]
lowdword_val]
```

<i>-u</i>	Show unreadable registers.
<i>begin-reg</i>	Starting register to display.
<i>end-reg</i>	Ending register to display.
<i>-w</i>	Write to a register.
<i>reg</i>	Identity of the register.
<i>hidword_val</i>	Write to the upper 32 bits of the register.
<i>lowdword_val</i>	Write to the lower 32 bits of the register.

Use

The MSR command is used to display or write to the Model Specific Registers. When no options are given to the MSR command, SoftICE will attempt to display all of these registers including the Value, Architectural Name, Read/Write flags, and Description. If you read a register that SoftICE does not have internal knowledge of, SoftICE will display the register number and its 64-bit value.

SoftICE has internal knowledge of the registers that make up the "Architectural MSRS" as defined in Volume 3 of the *Pentium 4 System Programming Guide* documentation from Intel.

Caution: Not all registers are available on all platforms. SoftICE makes no attempt to validate MSR writing. Writing to the **Model Specific Registers** has the potential to be very dangerous. Be certain of the destination and the value that you are writing.

Example

The following examples show the output from the MSR command.

In the first example, SoftICE displays all MSR registers from 0x10 through 0x20:

MSR 0x10 0x20

Reg	Value	Acc	ID	Name	Description
10	00000028:E03DB054	RW		IA32_TIME_STAMP_CTR	Time Stamp Counter
17	00000028:E03DB054	RW			
18	00000000:00000000				
1B	00000000:FEE00100	RW		IA32_APIC_BASE	APIC Location and Status

This time SoftICE displays non-readable registers. This is useful for finding undocumented Model Specific Registers.

MSR -u 0x10 0x17

Reg	Value	Acc	ID	Name	Description
10	0000002C:3B6C9262	RW		IA32_TIME_STAMP_CTR	Time Stamp Counter
NTICE: Error reading MSR 0x11					
NTICE: Error reading MSR 0x12					
NTICE: Error reading MSR 0x13					
NTICE: Error reading MSR 0x14					
NTICE: Error reading MSR 0x15					
NTICE: Error reading MSR 0x16					
17	20410000:00000000	R		IA32_PLATFORM_ID	Platform ID

In the final example, the Time Stamp Counter Register is read and then reset.

Note: This action causes the SoftICE blinking cursor to stop functioning until a popdown.

MSR 10

Reg	Value	Acc	ID	Name	Description
10	0000002D:62496928	RW		IA32_TIME_STAMP_CTR	Time Stamp Counter
MSR -w 10 0					
MSR 10					
Reg	Value	Acc	ID	Name	Description
10	00000000:2E350910	RW		IA32_TIME_STAMP_CTR	Time Stamp Counter

NAME

Type

Symbol/Source Commands

Definition

Assign a name to an address in memory.

Syntax

NAME [-d] [address] [name]

<i>-d</i>	Delete the specified name.
<i>address</i>	32-bit address or selector address.
<i>name</i>	The name to assign to the address.

Use

The NAME command is used to assign a name to an address. Assigned names will be displayed by SoftICE in the data display and disassembly windows, and can also be resolved by the expression evaluator.

User-defined names can be used to set “bookmarks” in code or data. You can name the first instruction of a disassembled routine for which you do not have source code, and return to the routine later using the U (unassemble) command. Similarly, a named data address can be displayed using any of the data display commands with the user-defined name.

Issuing the NAME command with no parameters will cause all of the defined names to be displayed. The NAME command with an address or name will display any existing names matching the given parameter. Wildcards are supported, so ‘NAME my*’ will display all defined names beginning with the characters ‘my’.

Although the NAME command allows selectors to be specified when defining a name (with the usual selector:offset format), selectors are ignored when searching for a name to match an address. In other words, the NAME command assumes a flat address model.

Names defined in application space – all addresses below 80000000h on most Windows platforms – are associated with the address context that was current when the name was defined. This means that names defined in application space will be local to a single application, and will not be

displayed in another application's address space, even though the actual address may be the same.

Names defined with this command will always use absolute addresses. If the module containing the named address is unloaded, and subsequently reloaded at a different address, all defined names will still point to the old, and now incorrect, addresses.

User-defined names co-exist with symbols defined in any loaded symbol or export tables. You can also set names within and around routines for which you have symbol names loaded. User-defined names do not show up if you are viewing code in source mode.

SoftICE allocates memory for the name command at startup. By default, 256 names are supported (the actual number of names you can set will be fewer if you use particularly long strings for your names), but the number of entries can also be controlled by adding a **NAMES= entry** to the **winice.dat** file. For example, adding **NAMES=400** to the winice.dat file will cause SoftICE to allocate memory for 400 names when it is started.

The NAME command is included on the popup mouse menu by default. To use this feature, simply right-click on an address anywhere in the SoftICE window, and select 'NAME' from the popup window. You will then need to complete the command by typing the name on the command line.

Example

Here is an example of setting a name on an address in the disassembly window. First the original disassembly window contents:

0008:805382E5	MOV	ESI,[EBP+08]
0008:805382E8	PUSH	EDI
0008:805382E9	JZ	805383AF

Now the enter the NAME command:

```
NAME 805383AF MysteryAddress1
```

The disassembled fragment now looks like this:

0008:805382E5	MOV	ESI,[EBP+08]
0008:805382E8	PUSH	EDI
0008:805382E9	JZ	MysteryAddress1

You can now enter

```
U MysteryAddress1
```

and SoftICE will disassemble the code starting at that address.

Note that setting a name on EBP+08 in the disassembly routine would succeed, but would not necessarily produce the desired result. This is because SoftICE would use the current value of EBP, whatever it happened to be at the time the command was issued. If the execution point was within the routine containing the code, and the stack frame had been set up, the name might well point at the correct address. If not, it could point anywhere. Use care when assigning names to addresses indexed with registers.

Finally, suppose you've created a number of names and now wish to delete them. Use

```
NAME -d *
```

to delete all defined names. To be more selective about the deletion, you might enter

```
NAME -d Mystery*
```

which would delete all names beginning with 'Mystery', leaving all others intact.

NET

Type

Customization

Definition

Remote debugging over standard IP ethernet connection.

Syntax

```
NET SETUP <IP-address | DHCP > [MASK=subnet-mask] [GATEWAY=IP-address]\ ALLOW <IP-address | ANY >] [PASSWORD=<password>]
```

IP-address IP-address of the machine on which you are running SoftICE.

DHCP Dynamic Host Configuration Protocol. Instructs SoftICE to get the IP parameters from your network DHCP server.

MASK=subnet-mask The network subnet mask.

GATEWAY=ip-address The IP address of the network gateway.

```
NET START <target-IP-address | DHCP > [MASK=subnet-mask] [GATEWAY=IP-address]
```

target-IP-address IP-address of the target machine on which you are running SoftICE.

DHCP Dynamic Host Configuration Protocol. Instructs SoftICE to get the IP parameters from your network DHCP server.

MASK=subnet-mask The network subnet mask.

GATEWAY=ip-address The IP address of the network gateway.

```
NET ALLOW <IP-address | ANY > [AUTO] [PASSWORD=<password>]
```

ANY Allows a machine from any IP address to connect to the target machine.

AUTO Allows a remote machine that has connected successfully to reconnect. This is useful if the remote machine loses its connection and must reconnect. If AUTO is not specified, you must reissue the NET ALLOW command on the target machine before another connection can be made.

PASSWORD=<password> A case-sensitive password that is required of users to get access to SoftICE on the target machine.

NET COMx [**baud-rate**]

x COM port number. Valid values are 1 through 4. This command uses only standard port addresses.

baud-rate Any legal baud rate between 1200 and 115200.

NET PING <*IP-address*>

NET RESET

NET STOP

NET HELP

NET STATUS

Use

You can use the NET commands to run SoftICE on one machine (the *target* machine,) so that it can be controlled remotely over a standard internet connection from another machine (the *remote* machine.) To run SoftICE remotely you must replace the adapter driver file on the target machine, with one provided in the SoftICE distribution. For details on setting up the target machine, refer to the chapter on “Customizing SoftICE” in the *Using SoftICE* document.

After installing the network adapter and driver, you can use the following NET command options on the target machine to enable SoftICE to be controlled by the remote machine.

The NET SETUP command enables the IP stack in SoftICE. This command identifies your IP parameters to SoftICE (IP-address, subnet-mask, and gateway address). If your local network supports DHCP (Dynamic Host Configuration Protocol), you can tell SoftICE to obtain the IP parameters from your network DHCP server. At this point, the IP stack is running, but SoftICE does not allow remote debugging until you use the NET ALLOW command to specify what other machine(s) can control SoftICE.

The NET START command enables the IP stack in SoftICE. This command identifies your IP parameters to SoftICE (IP-address, subnet-mask, and gateway address). If your local network supports DHCP (Dynamic Host Configuration Protocol), you can tell SoftICE to obtain the IP parameters from your network DHCP server. At this point, the IP stack is running, but SoftICE does not allow remote debugging until you use the NET ALLOW command to specify what other machine(s) can control SoftICE.

The NET ALLOW command tells SoftICE how to determine which machine(s) can be used to remotely control SoftICE. A remote machine can be specified as a specific IP address or ANY IP address. Access to control SoftICE can also be qualified by a case-sensitive password.

The NET COMx command allows you to establish a remote debugging session over a serial connection, including the ability to control the mouse and SoftICE window remotely.

The NET PING command allows you to do a basic network connectivity test by sending an ICMP Echo Request (PING) packet to an IP address. SoftICE sends the request and indicates whether it receives a response within 4 seconds.

The NET RESET command terminates any active remote debugging session (IP or serial connection) and cancels the effect of the previous NET ALLOW command.

The NET STOP command terminates any active remote debugging session (IP or serial connection) and cancels the effect of the previous NET ALLOW command. It also disables the IP stack.

The NET HELP command shows a list of the available network commands with their respective syntax.

The NET STATUS command shows the current status of the network adapter. It displays the current IP parameters (IP address, subnet mask, and gateway) and the status of the remote debugging connection.

After the target machine is set up correctly, you issue the SIREMOTE command on the remote machine to create the connection to the target machine. To use SIREMOTE, you must copy SIREMOTE.EXE from the SoftICE installation directory to the remote machine. The syntax for the SIREMOTE command is:

```
SIREMOTE <target-IP-address> [password]
```

The syntax for the SIREMOTE command with a serial connection is:

```
SIREMOTE COMx [<baud-rate>]
```

For more information about the SIREMOTE command, refer to the chapter on “Remote Debugging with SoftICE” in the *Using SoftICE* document.

Examples

The following commands set up SoftICE on a target machine whose IP address is 10.0.0.5, and allows a remote machine whose IP address is 10.0.0.10 to connect to the target machine and control SoftICE.

```
NET SETUP 10.0.0.5
NET START 10.0.0.5
NET ALLOW 10.0.0.10
```

Since the NET ALLOW command does not include the AUTO option, the remote user will only be allowed to connect to the target once.

The following commands set up SoftICE on a target machine that gets its IP address from a DHCP server. It then allows any other machine to connect if the user has the right password.

```
NET SETUP DHCP
NET START DHCP
NET ALLOW ANY AUTO PASSWORD=NuMega
```

This NET ALLOW command specifies that any IP address can be used to connect to the target machine if the user provides the proper password. The AUTO option tells SoftICE to allow remote machines to connect more than once in case of disconnect.

See Also

For more information on remote debugging, see the *Using SoftICE* document.

NTCALL

Type

System Information

Definition

Display NTOSKRNL calls used by NTDLL.

Syntax

NTCALL

Use

The NTCALL command displays all NTOSKRNL calls that are used by NTDLL. Many of the APIs in NTDLL are nothing more than wrappers 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 to 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 247.

Output

The NTCALL command displays all the level 0 APIs available. For each API, the following information displays:

<i>Func.</i>	Hexadecimal index number of the function passed in EAX.
<i>Address</i>	Selector:offset address of the start of the function.
<i>Params</i>	Number of dword parameters passed to the function.
<i>Name</i>	Either the symbolic name of the function, or the offset within NTOSKRNL if no symbols are loaded.

The following example shows the disassembler output. Note how SoftICE indicates that the INT 2Eh instruction's execution result in the NTOSKRNL function, _NtSetEvent being called with two 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.

NTCALL			
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

NTSTATUS

Type

System Information

Definition

Display header-defined mnemonics for NTSTATUS error codes.

Syntax

NTSTATUS *code*

Use

The NTSTATUS command displays the header-defined mnemonic associated with a specific NTSTATUS code. Many APIs in the operating system (especially the DDK) return NTSTATUS standard error codes. This command allows you to return the more intuitive mnemonic associated with any NTSTATUS error code.

Example

The following example shows the NTSTATUS command returning the mnemonic for the error code 0x5e:

```
NTSTATUS 0x5e  
OBJECT_INITIALIZATION_FAILED
```

O

Type

I/O Port

Definition

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 outputs are sent 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

The following command performs an out to Port 21, which unmask all interrupts for Interrupt Controller 1.

```
O 21 0
```

OBJDIR

Type

System Information

Definition

Display objects in the Windows Object Manager's object directory.

Syntax

```
OBJDIR [object-directory-name]
```

object-directory-name Name of the object as it appears in the Object Manager's object directory.

Use

Use the OBJDIR command to display the named objects within the Object Manager's object directory. Using OBJDIR without 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:

<i>Object</i>	Address of the object body.
<i>ObjHdr</i>	Address of the object header.
<i>Name</i>	Name of the object.
<i>Type</i>	Windows-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

Type

System Information

Definition

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:

Object-Type Name	Description
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.
CPD	Call Proc Data thunk.
ACCL	Accelerator table.
WSTN	Workstation 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:

<i>Object</i>	Pointer to the object's data.
<i>Type</i>	Type of the object.
<i>Id</i>	Object's type ID.
<i>Handle</i>	Win32 handle value for the object.
<i>Owner</i>	CSRSS specific instance data for the process or thread that owns the object.
<i>Flags</i>	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

OPINFO

Type

Miscellaneous

Definition

Display information about an assembly-language instruction.

Syntax

`OPINFO asm-instruction`

asm-instruction Assembly-language instruction.

Use

The OPINFO command displays a brief description of an assembly-language instruction. For general-purpose instructions, this command will also display a list of the affected flags in the EFLAGS register. OPINFO supports all x86 instructions, including SSE, SSE2, and AMD 3DNow! instructions. The x87 floating-point instructions are not supported.

The list of affected EFLAGS are shown as a table with an entry for each flag in the low 16-bits of EFLAGS. The possible values for each flag are:

Value	Meaning
<i>T</i>	Instruction tests flag.
<i>M</i>	Instruction modifies flag.
<i>O</i>	Instruction resets flag.
<i>1</i>	Instruction sets flag.
<i>-</i>	Instruction effect on flag undefined.
<i>R</i>	Instruction restores prior value of flag.
Blank	Instruction does not affect flag.

Example

The following example shows the OPINFO command used to display information about the ADC assembly-language instruction.

```
OPINFO adc
```

```
ADC
```

```
Integer addition: DEST <- DEST + SRC
```

EFLAGS		OF	DF	IF	SF	ZF	AF	PF	CF	TF	NT	RF	
		M			M	M	M	M	TM				

P

Type

Flow Control

Key

F10 (P RET: F12)

Definition

Execute one program step.

Syntax

P [RET]

<i>RET</i>	Return. Step until a return or return from interrupt instruction is found.
------------	--

Use

The P command executes 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, the P command uses a one-time INT 3 instruction execution breakpoint.

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 display with the bold video attribute. For call instructions, the highlighted registers 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.

The P command, by default, is thread specific. If the current CS:EIP is executing in thread X, SoftICE will not break until the program step occurs in thread X. This prevents the case of 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

The following example executes one program step.

P

PACKET

Type

System Information

Definition

Display the contents of a network packet.

Syntax

PACKET [*address*]

PACKET [*address*] [*length*]

PACKET ETHERNET | TOKEN-RING | ARCNET | FDDI

PACKET LINE | DETAIL | STRUCTURE

PACKET RAW | STANDARD

PACKET VALIDATE | NOVALIDATE

PACKET HELP

address Address of the network packet.

length Length of the network packet.

ETHERNET / TOKEN-RING / ARCNET / FDDI
Specifies a packet type.

LINE / DETAIL / STRUCTURE
LINE displays one line per packet; DETAIL displays detailed information per packet; STRUCTURE produces a structured element dump.

RAW / STANDARD RAW displays data in hexadecimal; STANDARD displays formatted/interpreted data.

VALIDATE / NOVALIDATE
VALIDATE turns “sanity checking” on; NOVALIDATE turns “sanity checking” off.

Use

Use the PACKET command to display the contents of a network packet.

Output

The output of the PACKET command varies depending on the options selected. See the example below.

Example

The following example shows the output of the PACKET command.

```
PACKET 85edf6e8
Ethernet:
DIX:
Destination: 00A0C98AB20A Size: 003C
Source: 00B0D02CEE87 Type: 0800
IP:
IP version: 4 IP header length: 5 (32-bit WORDs)
Type of service: 00
Precedence = Routine
Delay = Normal
Throughput = Normal
Reliability = Normal
Packet length: 003C Packet ID: 9BF5
More fragments: NO Fragment offset: 0000
Time-to-live: 128 Protocol: ICMP(1) Header checksum: 9CC9
(GOOD)
Source host id: 172.23.100.153
Destination host id: 192.80.49.1
ICMP:
Type = Echo(8)
Code = 0
Checksum = 5C0E
Identifier = 512
Sequence Number = 15616
ICMP Data:
===== Start of Text=====
abcdefghijklmnopqrstuvwabcdefghi
===== End of Text =====
```

PAGE

Type

System Information

Definition

Display page table information.

Syntax

PAGE [*address* [*l length*]]

<i>address</i>	The virtual address, segment:offset address, or selector:offset address about which you want to know page table information. The output includes the virtual and physical address.
<i>l length</i>	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.

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 4MB of memory (4KB/page * 1024 entries), and the page directory maps up to 4GB of memory (4MB/page table * 1024 entries).

The Windows NT family uses the 4 MB page feature of the Intel Pentium processors. NTOSKRNL, HAL, and all boot drivers are mapped into a 4 MB page starting at 2 GB (80000000h).

Note: In the above paragraph, we assume that the user space is 2 GB. That may not be the case; in some versions of NT Advanced Server and some Windows 2000/Windows XP platforms, the user space could be 3 GB.

When you specify the address parameter, 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, whether 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.

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, if fewer entries are listed than you specified, you must use a second PAGE command to list the pages starting where the first listing stopped.

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.

attributes

These are the page directory or page table entry attributes:

Value	Description
P	Present.
D	Dirty.
S	Supervisor.
RW	Read/Write.
A	Accessed.

Value	Description (Continued)
<i>U</i>	User.
<i>R</i>	Read Only.
<i>NP</i>	Not Present.
<i>4M</i>	4 MB page.

linear address range If the page directory is being displayed, then this is the virtual address of the page table. This is the address that you would use 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.

System	Private
Instance	Relock
VM	Hooked

If a page is marked **NP (Not Present)**, then all that is displayed is “NP” followed by the dword contents of the page table entry.

Example

When the Page command displays information on either PTEs or PDEs, 4 MB pages are indicated by the mnemonic “4M” in the Attributes column. 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 - 803FFFFFF				
00400000	P A S RW 4M	80400000 - 807FFFFFF				
00800000	P A S RW 4M	80800000 - 80BFFFFFF				
00C00000	P A S RW 4M	80C00000 - 80FFFFFF				
01034000	P A S RW 4M	81000000 - 813FFFFFF				

The following example shows how to use the PAGE command to display the attributes and addresses of the page from which instructions are currently being executed.

PAGE eip		
Linear	Physical	Attributes
80404292	00404292	P D A S RW

PAGEIN

Type

Category

Definition

Force a page of memory to be loaded into physical memory.

Caution: PAGEIN can be an unsafe command and is recommended for OS experts only. If the currently executing thread is not in a context in which it can touch pageable memory, issuing PAGEIN can crash your system. You should be sure you understand the state of your application and the effect of this command before attempting to use it.

Syntax

PAGEIN *address*

address Linear address of the page to be loaded.

Use

In some cases, a SoftICE command cannot retrieve the data you request because the actual physical memory that backs a particular linear address has been paged out by the operating system and so is not present in memory. You can use the PAGEIN command to force the page to be brought in from disk memory into physical memory.

Example

The following example shows the use of the PAGEIN command to load a page into physical memory.

```
PAGEIN 401000
```

See Also

PAGE

PAUSE

Type

Customization

Definition

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 PAUSE is off, the information automatically scrolls to the end of the command output.

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

Type

System Information

Definition

Dump the configuration registers for each PCI device in the system.

Syntax

```
PCI [-raw | -extended | -terse] [-b | -w | -d]  
    [bus][device][function]
```

<i>-raw</i>	Dumps the first 0x40 bytes of each function's PCI space.
<i>-extended</i>	Dumps all 256 bytes of each function's PCI space.
<i>-terse</i>	Dumps terse information that includes bus, device, and function information as well as device and vendor IDs.
<i>-b / -w / -d</i>	Information is dumped in byte word doubleword format.
<i>bus</i>	Bus number.
<i>device</i>	Device number.
<i>function</i>	Function number.

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 examples illustrate a part of the output for the PCI command.

```
PCI -extended 0 1f 0

Bus 00 Device 1f Function 00
Vendor: 8086 INTEL CORP
Device: 2410
Revision: 02
Device class: 06 Bridge device
Device subclass: 01 ISA bridge
Device sub-subclass: 00
Interrupt line: 00 Interrupt pin: 00 Min_Gnt: 00 MaxLat: 00
Cache line size: 00 Latency timer: 00 Header type: 80 BIST:
00
I/O:1 Mem:1 BusMast:1 Special:1 MemInv:0
Parity:0 Wait:0 SERR:0 Back2Back:0 Snoop:0

40: 00000801 00000010 00000000 00000000
50: 00000000 00000000 00000881 00000010
60: 09098A09 00000090 00000000 00000000
90: 0000FCFF 00000000 00000000 00000000
A0: 00000220 00000000 00000000 00000000
C0: 00000000 00000804 00000000 00000001
D0: 00002006 00000F02 00000000 00000000
E0: C0000010 140F0C01 00112233 00000771
F0: 00600000 00000000 00000F3A 00000000
```

```
PCI -terse

00/00/00 8086-7124 INTEL CORP
00/01/00 8086-7125 INTEL CORP
00/1E/00 8086-2418 INTEL CORP
00/1F/00 8086-2410 INTEL CORP
00/1F/01 8086-2411 INTEL CORP
00/1F/02 8086-2412 INTEL CORP
00/1F/03 8086-2413 INTEL CORP
01/07/00 1274-1371 ENSONIQ AudioPCI
01/0C/00 10B7-9200 3COM CORP 3Com EtherLink 10/100 PCI
NIC (3C905C-TX)
```

PEEK

Type

Display/Change Memory

Definition

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

Type

System Information

Definition

Display all virtual addresses that correspond to a physical address.

Syntax

PHYS *physical-address*

<i>physical-address</i>	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.

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. The

following example shows three different virtual addresses that correspond to physical A0000.

```
PHYS a0000  
000A0000  
004A0000  
80CA0000
```


POKE

Type

Display/Change Memory

Definition

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

Type

Customization

Definition

Print the contents of your SoftICE screen to a printer connected directly to a COM or LPT port.

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 only works with printers connected directly to a COM or LPT port. It does not work on network printers.

If you do not want to dump the information directly to a printer, you can save the SoftICE history buffer to a file. In the SoftICE Symbol Loader, choose SAVE SOFTICE HISTORY AS . . . from the File menu. For more information, see *Using SoftICE*.

From an NTVDM, use the DLOG.EXE utility to log the SoftICE Command window information.

See Also

PRN

PRN

Type

Customization

Definition

Set printer output port.

Syntax

```
PRN [lptx | comx]
```

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

The following command causes Print Screen output to go to the COM1 port.

```
PRN com1
```

See Also

Print Screen Key

PROC

Type

System Information

Definition

Display summary information about any or all processes in the system.

Syntax

```
PROC [-xom] [process-type | thread-type]
```

<i>-x</i>	Display extended information for each thread.
<i>-o</i>	Display list of objects in processes handle table.
<i>-m</i>	Display information about the memory usage of a process.
<i>process-type</i>	Process handle, process ID, or process name.
<i>thread-type</i>	Thread handle or thread ID.

Use

If you use the PROC command without any options, summary information is presented for the process you specify or, if none is specified, for all processes in the system. The information the memory option (-m) provides is also included when you specify the extended option (-x). The memory information 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 (-o) displays the object pointer, the object handle, and the object type for every object in the processes object handle table. Since object information is allocated from the system's pageable pool, the object's type name will not be available if the page is not present. In this case, question marks (???) are displayed.

Output

For each process the following summary information is provided:

<i>Process</i>	Process name.
<i>KPEB</i>	Address of the Kernel Process Environment Block.

<i>PID</i>	Process ID.
<i>Threads</i>	Number of threads the process owns.
<i>Priority</i>	Base priority of the process .
<i>User Time</i>	Relative amount of time the process spent executing code at user level.
<i>Krnl Time</i>	Relative amount of time the process spent executing code at the kernel level.
<i>Status</i>	Current status of the process: <ul style="list-style-type: none"> ◆ 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

In this example, the PROC command is used without parameters to list all the processes in the system. Note that the process that was active when SoftICE was popped up is highlighted, while the curently-active process/address context within SoftICE is indicated by an asterisk (*).

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
spoolsv	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

The following example uses the extended option (-x) to display extended information about a specific process, 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 HUPPEB: 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
```

PTR

Type

Display or Change Memory

Definition

Display a symbolic name for each pointer in a block of memory.

Syntax

```
PTR address [-i] [-r] [-s size] [-c count] [-n maxlookups]
```

PTR

<i>address</i>	Define the starting address for the memory block.
<i>-i</i>	Ignore addresses that have no known symbolic name.
<i>-r</i>	Reverse memory scan, decrementing address by size.
<i>-s size</i>	Define the size to increment or decrement the address by. Default is 4. Values can also be entered as b, w, d, or q.
<i>-c count</i>	Number of entries to display. Default is 10.
<i>-n maxlookups</i>	Define the maximum number of lookups per name. Default is 1.

Use

Use the PTR command to display the associated symbolic names for a block of memory locations. This is particularly useful when looking at the stack.

You are able to specify the number of entries to decode, increment size, and number of maximum symbolic lookups for each probed memory location. You can also specify reverse memory lookups using the *-r* option, or use the *-i* option to specify SoftICE should skip memory locations with no meaningful symbolic description. If you are attempting to track down a crash, and you do not know the passed parameters, the *-i* option is very useful. If you know the parameters being passed it is usually more helpful if you do not use the *-i* option. If you use the PTR command without any parameters, it will continue where the last PTR command ended (using the same parameter set).

Note: If you are using the *-i* option, the ending address will be equal to $\text{startaddress} - (\text{cnt} * \text{increment_amount})$. Without the *-i* option, the ending address will be equal to $\text{startaddress} + (\text{cnt} * \text{increment_amount})$.

Examples

The following example does a lookup for the first 10 addresses on the stack. Since the stack grows from higher to lower addresses, the actual stack is the opposite of what is listed. One thing to note with this example is that along with the return address you also have the parameter information.

```
:ptr esp
Start Address=80865548 End Address=8086556C Increment=4
0x80865548: 0x8086DDA0 is (a) _KiIdleThread0
0x8086554C: 0xFFDFFC50
0x80865550: 0xFFDFF980
0x80865554: 0x8085D70C is (a) @KiIdleLoop+0010
0x80865558: 0x0
0x8086555C: 0xE
0x80865560: 0x0
0x80865564: 0x0
0x80865568: 0x0
0x8086556C: 0x0
```

The following example shows the stack frame based off of EBP for the same function. This time the analysis is done in reverse order, as specified by the *-r* option and skips all unknown blocks. The purpose of this example is to find out as much information as possible about called functions.

```
:ptr ebp -r -i
Start Address=80865550 End Address=8086552C Increment=4
0x80865548: 0x8086DDA0 is (a) _KiIdleThread0
0x8086553C: 0x80843FE1 is (a) @PopIdle0+0005
0x80865538: 0xF0FE1D6C is (a) Int01+03A3
0x80865534: 0xF0FE28DD is (a) PopUp+0082
0x80865530: 0x80865550 is (a) _KiDoubleFaultStack+2D50
0x80865524: 0xFFDFF000 is (a) Processor Control Region
0x80865520: 0x80865534 is (a) _KiDoubleFaultStack+2D34
0x8086551C: 0x80865550 is (a) _KiDoubleFaultStack+2D50
0x80865518: 0x8086DDA0 is (a) _KiIdleThread0
0x80865514: 0x8086E000 is (a) Kernel Process (KPEB) for Idle(00)
```


The following example is similar to the second example, except it presumes that in analyzing the results you determined that a few of the items listed were not actually functions, but something else. You then enhanced the analysis by including multiple lookups per address, and have found that a few of the parameters at that location are actually IRPs.

```
:ptr ebp -r -i -n 3
Start Address=80865550 End Address=8086552C Increment=4
0x80865548: 0x8086DDA0 is (a) _KiIdleThread0
                is (a) IRP
0x8086553C: 0x80843FE1 is (a) @PopIdle0+0005
0x80865538: 0xF0FE1D6C is (a) Int01+03A3
0x80865534: 0xF0FE28DD is (a) PopUp+0082
0x80865530: 0x80865550 is (a) _KiDoubleFaultStack+2D50
0x80865524: 0xFFDFF000 is (a) Processor Control Region
0x80865520: 0x80865534 is (a) _KiDoubleFaultStack+2D34
0x8086551C: 0x80865550 is (a) _KiDoubleFaultStack+2D50
0x80865518: 0x8086DDA0 is (a) _KiIdleThread0
                is (a) IRP
0x80865514: 0x8086E000 is (a) Kernel Process (KPEB) for Idle(00)
                is (a) _KiIdleProcess
```

QUERY

Type

System Information

Definition

Display the virtual address space map of a process.

Syntax

`QUERY [[-x] address] [process-type]`

<i>-x</i>	Shows the mapping for a specific linear address within every context where it is valid.
<i>address</i>	Linear address to query.
<i>process-type</i>	Expression that can be interpreted as a process.

Use

The QUERY command displays a map of the virtual address space for a single process, 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

The QUERY command displays the following information:

<i>Context</i>	Address context.
<i>Address Range</i>	Start and end address of the linear range.
<i>Flags</i>	Flags from the node structure.
<i>MMCI</i>	Pointer to the memory management structure.
<i>PTE</i>	Structure that contains the ProtoPTEs for the address range.
<i>Name</i>	Additional information about the range. This includes the following: <ul style="list-style-type: none">◆ Memory-mapped files that will show the name of the mapped file.◆ Executable modules that will show the file name of the DLL or EXE.

- ◆ Stacks that will be displayed as TID (thread ID).
- ◆ Thread information blocks that will be displayed as TID.
- ◆ Any address that the WHAT command can identify.

Example

The following example uses the QUERY command to list the address map of the PROGMAN process.

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	rpccltcl.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

Type

Display/Change Memory

Definition

Display or change the register values.

Syntax

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

The following example sets the AH register equal to 5.

```
R ah=5
```

The following example toggles the O, Z, and P flag values.

```
R fl=ozp
```

The following example moves the cursor into the Register window position under the first flag field.

```
R fl
```

The following 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

Type

Window Control

Key

F4

Definition

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

Type

Miscellaneous

Definition

Search memory for data.

Syntax

S [-acu][*address l length data-string*]

<i>-a</i>	Specifies “find all matches in search range” as opposed to “find the first match in the search range and stop.”
<i>-c</i>	Makes search case-insensitive.
<i>-u</i>	Searches for UNICODE string.
<i>address</i>	Starting address for search.
<i>l</i>	length Length in bytes.
<i>data-string</i>	List of bytes or quoted strings separated by commas or spaces. A quoted string can be enclosed with single or double quotes.

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 (10h).

Example

The following 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
```

The following example searches the entire 4GB virtual address range for 'string.'

```
S 30:0 L ffffffff 'string'
```


SERIAL

Type

Customization

Definition

Redirect console to serial terminal.

Note: The SERIAL command was used for the Windows 3.1 and Windows 9.x operating systems and is no longer supported. Information is supplied here for informational purposes only. The SERIAL command functions are now provided through the NET command.

Syntax

```
SERIAL [on | VT100 | [com-port] [baud-rate] | off]
```

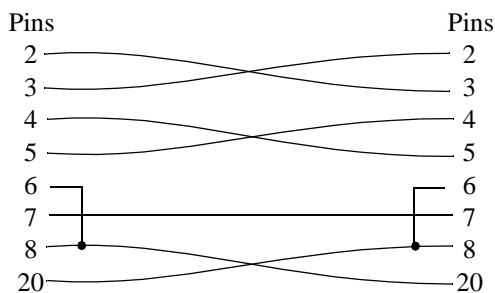
<i>VT100</i>	Initiates VT100 serial mode.
<i>com-port</i>	Number from 1 to 4 that corresponds to COM1, COM2, COM3 or COM4. Default is COM1.
<i>baud-rate</i>	Baud-rate to use for serial communications. The default is to have SoftICE automatically determine the fastest possible baud-rate that can be used. The rates are 1200, 2400, 4800, 9600, 19200, 23040, 28800, 38400, 57000, 115200.

Use

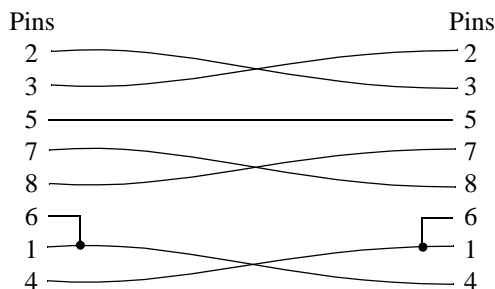
Use the SERIAL command to establish a remote debugging session through a serial port. Refer to *DIAL* on page 68 for information about how to establish remote sessions over a modem, and to the chapter on “Remote Debugging with SoftICE” in the *Using SoftICE* document for a detailed explanation of this procedure.

Remote debugging requires a second IBM-compatible PC. 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.



25-Pin Null-Modem Configuration



9-Pin Null-Modem Configuration

Before using the SERIAL command on the local machine, you must first run the SERIAL32.EXE or SERIAL.EXE program on the remote machine. SERIAL32.EXE is a 32-bit client. SERIAL.EXE is a real mode (MS-DOS) client. The syntax for the SERIAL32.EXE and SERIAL.EXE programs are the same as the syntax of the SERIAL command, so the following information is applicable to all.

The SERIAL command has two optional parameters. The first parameter specifies the comport on the machine where the command is entered through which the connection will be made. If no comport is specified, Comport 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.

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. For example, the following keyword sets Com 2 as the serial post.

```
Com2
```

For Windows 9x

Select the desired comport in the remote debugging initialization settings within Symbol Loader.

Example

The following example shows how to run the SERIAL.EXE program on the remote machine:

```
SERIAL.EXE on 19200
```

The following example shows how to execute a SERIAL command on the local machine that corresponds to the SERIAL.EXE command given in the previous example.

```
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

ANSWER; DIAL; NET

SET

Type

Mode Control

Definition

Display or change the state of an internal variable.

Syntax

SET [*keyword*] [*on* | *off*] [*value*]

- keyword* Specifies option to be set.
- on, off* Enables or disables the option.
- value* Value to be assigned to the option.

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. The SoftICE TAB-AutoCompletion feature can be used to finish partially-typed keywords in the SET command. Press TAB after entering a partial keyword, and SoftICE will attempt to complete the keyword for you. SET supports the following keywords:

Keyword	Options
ALTSCR	[mono vga off]
ASSERT	[on off]
BUTTONREVERSE	[on off]
CASESENSITIVE	[on off]
CENTERING	[on off]
CHECKSTRINGS	[on off]
CODE	[on off]
DISASSEMBLYHINTS	[on off]

Keyword	Options (Continued)
EXCLUDE	[on off]
FAULTS	[on off]
FLASH	[on off]
FONT	[1 2 3]
FORCEPALETTE	[on off]
I1HERE	[on off]
I3HERE	[on off]
LONGTYPENAMES	[on off]
LOWERCASE	[on off]
MAXIMIZE	[on off]
MONITOR	[0 1 2 3 <i>n</i>]
MOUSE	[on off] [1 2 3]
ORIGIN	x y (window location in pixel coordinates)
PAUSE	[on off]
REFERENCE	[on off]
SELECTORS	[on off]
SYMBOLS	[on off]
TABS	[on off] [1 2 3 4 5 6 7 8]
THREADP	[on off]
TYPEFORMAT	[1 2 3]
VERBOSE	[on off]
WHEELINES	<i>n</i>

SET ALTSCR MONO will redirect the SoftICE output to an alternate monochrome monitor using a Hercules-compatible monochrome card. This mode displays 43 lines of text rather than the 25 lines displayed in text mode. You can also go to the SoftICE program group and use the Display Adapter Setup to select the monochrome monitor. Then, SoftICE will automatically start out in monochrome mode making the ALTSCR command unnecessary. You should also use this setting if you are experiencing video problems.

SET ALTSCR VGA will redirect the SoftICE output to an alternate monitor using standard VGA mode. ALTSCR VGA users should note that if you use an alternate screen in VGA mode, you must disable VGA on the graphics card that will be used to display Windows. You cannot use two cards that are in VGA mode at the same time. Consult the documentation for your graphics card to find the appropriate PCI slot or switches to set.

SET ALTSCR OFF will turn off the SoftICE output to an alternate monitor. The default setting is **ALTSCR OFF**.

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 the chapter on “Customizing SoftICE” in the *Using SoftICE* guide.

SET ASSERT OFF will prevent SoftICE from popping up on RtlAsserts. Text will still be displayed to the command window and can be viewed on the next popup. If **SET ASSERT ON** is selected, SoftICE will pop up on each RtlAssert.

SET BUTTONREVERSE ON reverses the meaning of the left and right mouse buttons.

SET CASESENSITIVE ON makes global and local symbol names case sensitive. Enter them exactly as displayed by the SYM command.

SET CENTERING ON centers the SoftICE window. When you manually move the window, SoftICE turns centering off.

SET CHECKSTRINGS ON makes the disassembler examine operands. If an operand appears to point to an ASCII or Unicode string, the disassembler will display the string as a comment.

SET CODE ON will display the machine instruction bytes.

SET DISASSEMBLYHINTS ON makes the disassembler display directional hints for branch instructions. If a disassembled instruction is a branch (conditional or unconditional), the disassembler will display a directional arrow next to the address operand, pointing towards the destination address.

SET EXCLUDE ON will cause SoftICE to check for exclusion addresses before accessing memory. **SET EXCLUDE OFF** will keep SoftICE from checking for exclusion addresses before accessing memory.

SET FAULTS OFF will prevent SoftICE from popping up on user mode faults.

SET FLASH ON will cause SoftICE to redraw its entire screen after every step and trace. Turn this on if you are debugging applications which render to the screen and overwrite the SoftICE display.

SET FONT n changes the font size when in Universal Display Mode. The font size is not alterable on VGA, remote debugging, or monochrome monitors.

SET FORCEPALETTE ON prevents the system colors (Palette Indices 0-7 and 248-255) from being changed in 8-bits per pixel mode. This ensures that the SoftICE display can always be seen. This is **OFF** by default.

SET I1HERE ON causes SoftICE to pop up on all instances of **Int 1**.

SET I3HERE ON causes SoftICE to pop up on all instances of **Int 3**. If the **DRV** option is given, only Int 3s that occur within the kernel space will cause SoftICE to pop up.

SET LONGTYPENAMES OFF turns off long typenames. When set to **ON**, "unsigned longs" will be shown as "ulong." All other types will be shortened accordingly.

SET LOWERCASE ON causes disassembly to be in lower case; if set to **OFF**, disassembly is in upper case.

SET MAXIMIZE ON sizes the UVD window as large as possible. This setting overrides **LINES** and **WIDTH** but not **FONT**. When you can change the **LINES** or **WIDTH** settings, SoftICE changes them only temporarily. The next time SoftICE pops up, it displays the window at maximum size.

SET MONITOR n changes the monitor used to display SoftICE. Enter the decimal value representing the monitor you want to use to display SoftICE. Issuing the **SET MONITOR** command without parameters lists the monitors available to SoftICE. If you do not want SoftICE to patch in a specific video driver, add the base name of the DDI driver to the **NTICE\ExcludedDisplayDrivers** key in the registry. This list is delimited by semi-colons (;).

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 ORIGIN *x y* allows you to control the location of the SoftICE window in the Universal Video Driver (UVD) mode. The *x* and *y* parameters represent the pixel coordinates for the top left corner of the SoftICE window. For example, to place the SoftICE window in the top left corner of the monitor screen, type **SET ORIGIN 1 1**. To view the current origin setting, use the **SET ORIGIN** command without any parameters.

SET PAUSE ON causes a prompt to appear in the status line at the bottom of the SoftICE window before information scrolls off the window. Use **Enter** to scroll a single line at a time. Use any other key to scroll a page at a time. **SET PAUSE OFF** causes the information to automatically scroll to the end of the command output. If you do not specify a parameter, the current state of PAUSE displays. The default is **SET PAUSE ON**.

SET REFERENCE ON causes the SoftICE code window to display symbol names when disassembling instructions, if they are available. **SET REFERENCE OFF** causes the code window to display only addresses when disassembling instructions. The default value is **SET REFERENCE ON**. To view the current REFERENCE setting, use the **SET REFERENCE** command without any parameters.

SET SELECTORS ON causes the selector value to be shown with each disassembly address. **SET SELECTORS OFF** causes the selector value to be shown in the Code Window's title bar only.

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.

SET TABS *n* changes the number of spaces to replace for each tab character.

SET THREADP OFF turns off thread-specific stepping within a process.

SET TYPEFORMAT *n* defines the layout and format of the locals window.

- ◆ **SET TYPEFORMAT 1** is <type> <variable name> = <value>.
- ◆ **SET TYPEFORMAT 2** is the default <variable name> <type> = <value>.
- ◆ **SET TYPEFORMAT 3** is <variable name> = <value> <type>. There's also a winice.dat variable to control this. (TYPEFORMAT valid values are 1, 2, and 3.)

SET VERBOSE OFF turns off the SoftICE information messages such as LOAD32, UNLOAD32, and EXIT32.

SET WHEELINES sets the number of lines that should be scrolled for each wheel movement when using an Intellipoint mouse.

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; THREAD

SRC

Type

Symbol/Source

Key

F3

Definition

Cycle among source, code, and mixed displays in the Code window.

Syntax

SRC

Use

Use the SRC command to cycle among the following modes in the Code window:

Tip: Use F3 to cycle modes quickly.

- ◆ Source Mode
- ◆ Code Mode
- ◆ Mixed Mode

Example

The following example changes the current mode of the Code window.

```
SRC
```

SS

Type

Symbol/Source

Definition

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 and proceeds to the end of the code. The search will not wrap back through the beginning of the code once the end is reached. 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

Type

System Information

Definition

Display a call stack.

Syntax

```
STACK [-v | -r] [task-name | thread-type | SS:EBP]
```

<i>-v</i>	Verbose. Displays local variables in 32-bit code.
<i>-r</i>	Ring transition. Walks through ring transitions in 32-bit code.
<i>task-name</i>	Name of task as displayed by the TASK command inside an NTVDM.
<i>thread-type</i>	Thread handle or thread ID.
<i>SS:EBP</i>	SS:EBP of a valid stack frame.

Use

Use the STACK command to display the call stacks for NTVDM programs, Windows tasks, and 32-bit code.

If you enter STACK with no parameters, the current SS:EBP is used as a base for the stack frame displayed. You can explicitly specify a stack base with a task-name, a base address, and 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:EBP of a valid 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:EBP 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. In 32-bit code, the -r (ring transition) switch tells SoftICE to continue walking the stack through ring transitions. The SoftICE stack walking algorithm can use FPO (frame pointer omission) data to walk call stacks. The FPO data is module/symbol table specific. Therefore, when using the STACK

command, it will be helpful to have symbol tables for all modules that are listed on the stack. If SoftICE does not have FPO data, it is limited to walking EBP frames only.

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 any loaded symbol table. If the routine is not in the symbol table, the export list and module name list are searched for nearby symbols.

In 32-bit code, the STACK command output includes the frame pointer, the return address, and the instruction pointer for each frame. If you set the -v switch, SoftICE also displays the local variables for each frame. 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 Windows device driver code at ring 0. The call stack is followed through ring transitions when you set the -r switch.

Example

The following example shows the output from the STACK -r command when sitting at a breakpoint in the DriverWorks PCIENUM sample. Using the -r parameter results in the STACK command walking past the ring transition in _KiSystemService. The output is organized into three columns. Column one is the frame pointer. Column two is the return address. Column three is the instruction pointer.

```
STACK -r
FC070DE8 F74FC919      KIrp::KIrp+0007
FC070E04 F74FC796      KDevice::DeviceIrpDispatch+003C
FC070E18 801FE4F8      KDriver::DriverIrpDispatch+0026
FC070E30 8016EBF8      @IoofCallDriver+0037
FC070E48 8016CDF7      _IopSynchronousServiceTail+006A
FC070ED8 8013DC14      _NtReadFile+0683
FC070F04 77F67E87      _KiSystemService+00C4
0012FE04 77F0D300      ntdll!.text+6E87
0012FE6C 100011A0      _ReadFile+01A6
0012FEDC 00401057      PCIDLL!.text+01A0
0012FF80 004017C9      PCIEXE!.text+0057
0012FFC0 77F1BA3C      PCIEXE!.text+07C9
0012FFF0 00000000      _BaseProcessStart+0040
```

The following example shows the same stack as the previous example, but displayed with the `STACK -v -r` command. The `-v` switch causes the local variables for each frame to be displayed.

```
STACK -v -r
F74AFDE8 F74FC919      KIrp::KIrp+0007
      [EBP-4] + const class KIrp * this = 0xF74AFDF4 <{...}>
      [EBP+8] +struct _IRP * pIrp = 0x84C460E8 <{...}>
F74AFE04 F74FC796      KDevice::DeviceIrpDispatch+003C
      [EBP-C] + const class KDevice * this = 0x807DC7A8 <{...}>
      [EBP-4] unsigned long Major = 0x3
      [EBP+8] +struct _IRP * pIrp = 0x84C460E8 <{...}>
F74AFE18 801FE4F8      KDriver::DriverIrpDispatch+0026
      [EBP+8] +struct _DEVICE_OBJECT * pSysDev = 0x807DB850
<{...}>
      [EBP+C] +struct _IRP * pIrp = 0x84C460E8 <{...}>
F74AFE30 8016EBF8      @IoofCallDriver+0037
F74AFE48 8016CDF7      _IopSynchronousServiceTail+006A
F74AFED8 8013DC14      _NtReadFile+0683
F74AFF04 77F67E87      _KiSystemService+00C4
0012FE04 77F0D300      ntdll!.text+6E87
0012FE6C 100011A0      _ReadFile+01A6
0012FEDC 00401057      PCIDLL!.text+01A0
0012FF80 004017C9      PCIEXE!.text+0057
0012FFC0 77F1BA3C      PCIEXE!.text+07C9
0012FFFF 00000000      _BaseProcessStart+0040
```

The following example shows the output of the STACK command in 16-bit mode. The command has been issued without any parameters, 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
```

Each entry of the call stack in the 16-bit format contains the following information:

- ◆ Symbol name or module name in which the return address falls
- ◆ SS:EBP 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:EBP relative offset
- ◆ Stack variable name
- ◆ Data in the stack variable if it is of type char, int, or long

SYM

Type

Symbol/Source

Definition

Display or set symbols.

Syntax

```
SYM [[segment*!]symbol*] [value]
```

segment A valid section name (such as .text) or a partial section name. You can use this parameter to display symbols in a particular section.

symbol A valid symbol name or partial symbol name.

value A value you want to set for the symbol.

Note: The SYM command supports regular expression searches.

Use

Use the SYM command to display and optionally set symbol addresses. If you enter SYM without parameters, all symbols in the current symbol table are displayed. The address of each symbol is displayed next to the symbol name.

If you specify a section name (such as .data or .text) followed by an exclamation point (!) and a symbol name or asterisk (*), SYM displays only symbols from the specified section.

The SYM command can also be used to search for a symbol when you can only remember a portion of the name. SYM will accept:

- ◆ simple wildcard searches where the partial name is followed by an asterisk (*)
- ◆ simple wildcard searches where commas (,) are used to indicate single unknown characters
- ◆ searches using complex regular expressions

To use a regular expression to search for symbols, the search patterns must be enclosed in slash (/) characters. If the search patterns contain spaces, the whole expression, including slashes, must also be enclosed in double-quotes ("). There may be two separate search patterns, one for the section, and one for the symbol name, and either or both may be regular expressions. The full syntax of regular expressions is beyond the scope of this manual, but some examples are given below, and references can be found on the Internet.

Search expressions are case-insensitive by default. You can use the `SET CASESENSITIVE` command to toggle the case sensitivity.

If a value is specified after the search pattern, the address of all symbols that match the pattern are set to the value.

Example

The following example displays all symbols that start with FOO.

```
SYM foo*
```

The same thing could also be done with a regular expression:

```
SYM /^foo/
```

The caret (^) at the beginning of this regular expression specifies that the next character must match the beginning of the symbol. Without the caret this pattern would match any symbol containing the characters FOO at any position in the name. Once you have found a FOO, this command could be used to change its address (this is rarely useful):

```
SYM foo* FB007500
```

The following example displays all sections for the current symbol table:

```
SYM !
```

The following two examples both display all symbols in section .data that start with FOO:

```
SYM main!foo*  
SYM main!/^foo/
```

The following final example has two regular expression search patterns: one for the section and one for the symbol. This will search all sections with names that start with “.data” followed by an optional digit, so “.data0” will also be searched, if it exists. The command will then look for symbols which contain the string “Apic” somewhere, followed by zero or more unknown characters, followed by the letter "V".

```
SYM /.data?[/!]/Apic.*V/
```

When run on HAL, this command finds _HalPIOApicVersion, _rgzApicNotVerified, and _rgzBadApicVersion, all in section .data.

SYMLOC

Type

Symbol/Source

Definition

Relocate the symbol base.

Syntax

`SYMLOC [segment | o | r | -c process-type |
 section selector base]`

<i>segment</i>	The segment address is only used to relocate NTVDM programs.
<i>o</i>	For 16-bit Windows table only. Changes all selector values back to their ordinal state.
<i>r</i>	For 16-bit Windows table only. Changes all segment ordinals to their appropriate selector value.
<i>-c</i>	Specify a context value for a symbol table. Use when debugging extended applications within NTVDM.
<i>process-type</i>	Specify a process.
<i>section</i>	For 32-bit tables only. PE file 1 based section-number.
<i>selector</i>	For 32-bit tables only. Protected mode selector.
<i>base</i>	For 32-bit tables only. Base linear address of the section.

Use

The SYMLOC command handles symbol fixups in a loaded symbol table. The command contains support for 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 NTVDM 32-bit extender applications.

In an NTVDM 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.

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 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.

```
SYMLOC 1244
```

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

Type

Flow Control

Key

F8

Definition

Trace one instruction.

Syntax

T [=address] [count]

=address Address at which to begin execution.

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

The following example single-steps through eight instructions starting at memory location CS:1112.

```
T = CS:1112 8
```

TABLE

Type

Symbol/Source

Definition

Change or display the current symbol table.

Syntax

```
TABLE [table-name | AUTOON | AUTOOFF | r originaltablename  
newtablename]
```

<i>table-name</i>	Symbol table name or enough of the first few characters to define a unique name.
<i>AUTOON</i>	Key word that turns auto table switching on.
<i>AUTOOFF</i>	Key word that turns auto table switching off.
<i>r</i>	Renames original <i>table-name</i> to new <i>table-name</i> .
<i>originaltablename</i>	Original table-name you want to change.
<i>newtablename</i>	New name to rename <i>table-name</i> as.

Note: Use **View Symbol Tables** in the SoftICE Loader to remove tables from memory.

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. The TABLE command also has its own TAB-AutoCompletion routine. You can hit TAB at any point while typing the table-name, and SoftICE will attempt to complete the name for you.

Use the TABLE command when you have multiple symbol tables loaded. SoftICE supports symbol tables for 16- and 32-bit Windows applications and DLLs, Windows NT family device drivers, and NTVDM programs.

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 switches to auto table switching mode. In this mode, SoftICE changes the current table to the 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.

Symbol tables can be tied to a single address context or multiple address contexts. If a table is tied to a single 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.

By using the *r* option, you can alter the internal mapping of symbol table names to match the module name as displayed with the **MOD** command. This is most typically used when the installed name of a module is different than the name under which it was built.

Example

In the following example, the TABLE command, used without parameters, lists all loaded symbol tables. In the sample output, `ntoskrnl` is highlighted because it is the current table. Also, `nmdemola` is out of date the symbol table has a different timestamp than the binary image. When this occurs it should be corrected by retranslating and reloading the symbol table. The amount of available symbol memory is displayed at the bottom.

```
TABLE
hal [NM32]
ntoskrnl [NM32]
*nmdemola [NM32]
0001040593 Bytes of Symbol Memory Available
WARNING: One or more symbol tables (*) are out of date!
```

In the following example, the current table is changed to `nmdemola.exe`. Notice that only enough characters to identify a unique table were entered. You can also use the SoftICE TAB-AutoCompletion to finish partial table names.

```
TABLE nm
```


TASK

Type

System Information

Definition

Display the Windows task list when the current context is that of an NTVDM process containing a WOW box.

Syntax

TASK

Use

The TASK command is process specific and only shows 16-bit tasks. In addition, it is only useful when the current context is that of an NTVDM process containing a WOW box. To view information on other processes, refer to *PROC* on page 212.

Output

For each running task, the following information displays:

<i>Task Name</i>	Name of the task.
<i>SS:SP</i>	Stack address of the task when it last relinquished control.
<i>StackTop</i>	Top of stack offset.
<i>StackBot</i>	Bottom of stack offset.
<i>StackLow</i>	Lowest value that SP has ever had when there was a context-switch away from the task.
<i>TaskDB</i>	Selector for the task data base segment.
<i>hQueue</i>	Queue handle for the task. This is just the selector for the queue.
<i>Events</i>	Number of outstanding events in the queue.

See Also

PROC

THREAD

Type

System Information

Definition

Display information about a thread.

Syntax

```
THREAD [-r | -x | -u | -w] [thread-type | process-type]
```

<i>-r</i>	Display value of the thread's registers.
<i>-x</i>	Display extended information for each thread.
<i>-u</i>	Display threads with user-level components.
<i>-w</i>	Display a list of the objects that the thread is waiting on.
<i>thread-type</i>	Thread handle or thread ID.
<i>process-type</i>	Process-handle, process-id or process-name.

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 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 the thread context switches: ESI, EDI, EBX and EBP.

Output

For each thread, the following summary information is displayed:

<i>TID</i>	Thread ID.
<i>Krnl TEB</i>	Kernel Thread Environment Block.
<i>StackBtm</i>	Address of the bottom of the thread's stack.
<i>StackTop</i>	Address of the start of the thread's stack.

<i>StackPtr</i>	Threads current stack pointer value.
<i>User TEB</i>	User thread environment block.
<i>Process(Id)</i>	Owner process-name and process-id.

When you specify extended output (-x), THREAD displays many fields of information about thread environment blocks. Most of these fields are self-explanatory, but the following are particularly useful and deserve to be highlighted:

<i>TID</i>	Thread ID.
<i>KTEB</i>	Kernel Thread Environment Block.
<i>Base Pri, Dyn. Pri</i>	Threads base priority and current priority.
<i>Mode</i>	Indicates whether the thread is executing in user or kernel mode.
<i>Switches</i>	Number of context switches made by the thread.
<i>Affinity</i>	Processor affinity mask of the thread. Bit positions that are set represent processors on which the thread is allowed to execute.
<i>Restart</i>	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)	

The following 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

WT

TIMER

Type

System Information

Definition

Display information about timer objects.

Syntax

TIMER [*timer-address*]

timer-address Location of a timer object.

Use

Displays the system timer objects or the contents of a specific timer object.

Example

The following example shows a portion of the output of the **TIMER** command when it is issued with no parameters.

TIMER					
Timer	DPC	DPC	Remaining		
Object	Address	Context	Time	Signaled	Period
Symbol					
80706588			10.233s	FALSE	
80681C48			10.233s	FALSE	
8074E108			62.787s	FALSE	
80730DE8			10.248s	FALSE	
FBDA3980	FBD47C80	00000000	18.588ms	FALSE	
NTice!.text+000479C0					
FC392EB0	F74D0B4C	FC392E80	19.884ms	FALSE	
TDI!.text+088C					
806DAD68			22.633ms	FALSE	
8066A108			29.323ms	FALSE	
807946D8	80802E90	807946A8	180.777s	FALSE	
807AF048			59.942ms	FALSE	
8078D1A8	80802EF0	8078D1A8	79.971ms	FALSE	
807079C8			5.223s	FALSE	
8074B108			68.043s	FALSE	
8073D108			159.510ms	FALSE	

The following example shows the output of **TIMER** when it is issued for a specific timer object.

```
TIMER 80793568
Timer Object at 80793568
Dispatcher Type: 08
Dispatcher Size: 010A
Signal State: Not Signaled
Dispatch Wait List Forward Link: <self>
Dispatch Wait List Back Link: <self>
Remaining Time: 349.784ms
Timer List Forward Link: 8014D828
Timer List Back Link: 8014D828
Timer Object is NOT Periodic
Timer DPC: 80793548
DPC Routine: F74D0B4C TDI!.text+088C
DPC Context: 80793538
```

See Also

APC; DPC

TSS

Type

System Information

Definition

Display task state segment and I/O port hooks.

Syntax

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:

<i>TSS selector value</i>	TSS selector number.
<i>selector base</i>	Linear address of the TSS.
<i>selector limit</i>	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

On Windows 9x and the Windows NT family, the TSS command also displays the I/O permission map base and size. 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.

```
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
```


TYPES

Type

Symbol/Source Command

Definition

List all types in the current context or list all type information for the specified type-name .

Syntax

TYPES [*type-name*]

type-name A type-name (or partial type-name) to display.

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 types that match the name. If only one type matches the name you specified, TYPES displays detailed type information. If the match is a structure, TYPES expands it and lists the type for each of its members.

The types command supports regular expressions. To use a regular expression to search for types, the search pattern must be enclosed in slash (/) characters. If the search patterns contain spaces, the whole expression, including slashes, must also be enclosed in double-quotes (").

Example

The following example displays all the types in the current context. The example output is only a partial listing.

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 type information for the type-name `_bouncedata`.

TYPES _bouncedata		
typedef struct _BOUNCEDATA {		
public:		
	void * hBall1 ;	
	void * hBall2 ;	
	void * hBall3 ;	
	void * hBall4 ;	
};		

The following example shows the use of a regular expression to search for matching types.

TYPES /_POOL.*TABLE/		
Size	Type Name	Typedef
0x001C	_POOL_TRACKER_TABLE	struct _POOL_TRACKER_TABLE

See Also

LOCALS; WL

U

Type

Display/Change Memory

Definition

Unassemble instructions.

Syntax

U [*address* [*L length*]]

<i>address</i>	Segment offset or selector offset.
<i>symbol-name</i>	Scrolls the Code window to the function you specify.
<i>length</i>	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. The Command window displays either eight lines 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, the U command displays the instruction using the reverse video attribute. If the current CS:EIP instruction is a relative jump, the instruction contains either the string JUMP or NO JUMP, indicating whether or not the jump will be taken. If the jump will be taken, an arrow indicates if the jump will go up or down in the Code window. If the current CS:EIP instruction references a memory location, the U command displays the contents of the memory

location in the Register window beneath the flags field. If the Register window is not visible, this value displays at 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.

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

The following command disassembles 100h bytes starting at MyProc and displays the output in the Command window.

```
U myproc L100
```

USB

Type

System Information

Definition

Displays information about USB host controllers installed in the system.

Syntax

USB [-dumpregs | -schedule[-v]] [host controller number]

- | | |
|-------------------------------|--|
| <i>-dumpregs</i> | Produces a detailed listing of the contents of the device's control registers. |
| <i>-schedule[-v]</i> | Displays the current contents of the USB transaction schedule for the specified host controller. This command also accepts a verbose switch (-v), which will cause it to display inactive entries as well. |
| <i>host controller number</i> | Specifies host controller number from the list generated by the USB command with no parameters. |

Use

The USB command displays information about the USB host controllers installed in the system. Issued with no parameters, the USB command will display a numbered list of the host controllers detected. The -dumpregs and -schedule switches both require the user to specify a host controller number from the list. USB **-dumpregs** [HC number] will show the contents of the control registers for the specified host controller, while USB **-schedule** [HC number] will display the current contents of the USB transaction schedule for the specified host controller.

The output produced by the USB command with no parameters will contain a numbered list of the USB controllers in the system, in the order they were detected by SoftICE. It will also display which of the host controller specifications the device complies with, UHCI, or Universal Host Controller Interface; or OHCI, the Open Host Controller Interface. (EHCI, currently the only host controller specification for USB 2.0, will be supported in a future release of SoftICE). Which specification the host controller supports will determine the output from the -dumpregs and -schedule options. The PCI address of each device is also listed, allowing the user to easily access more information about the device using the PCI command.

The **-dumpregs switch** will produce a detailed listing of the contents of the device's control registers. Host controller registers are defined by the host controller specification the device conforms to, so the output from this command will differ depending on the HC type. The user will need to consult the host controller specifications themselves for a detailed description of the various control registers.

The **-schedule switch** produces a list of the currently active entries in the host controller's schedule. This command also accepts a **verbose switch (-v)**, which will cause it to display inactive entries as well. For UHCI controllers the entire schedule is displayed, but for OHCI controllers this command displays only the interrupt entries in the schedule; bulk and isochronous transactions are not shown.

Note: If SoftICE is set up to use a USB keyboard or mouse when the USB -schedule command is issued, you may see SoftICE's own entries in the USB schedule, rather than Windows'. This is because SoftICE patches the USB schedule when it is popped up, in order to use the keyboard and mouse. The patching will only affect USB keyboard and mouse devices, not all USB devices in the system. If you need to see the USB schedule with Windows' keyboard and mouse schedule entries intact, you should disable SoftICE's USB input device support using the Troubleshooting tab in the DriverStudio Configuration (Settings) dialog.

Example

This is the output from the USB command on a particular system.

```
USB
3 USB Host Controllers Found
HC 0: UHCI at PCI Bus 0 Device 1F Function 2
HC 1: UHCI at PCI Bus 0 Device 1F Function 4
HC 2: OHCI at PCI Bus 4 Device F Function 0
```

Here is the output from the `-dumpregs` command, shown on a UHCI controller:

```
USB -du 0
USB I/O registers for Host Controller 00:
Universal Host Controller at PCI Bus 00 Device 1F Function 02
USB Command (FF80) = 0081
    MaxP:64Bytes CF:0 SWDBG:0 FGR:0 EGSM:0 GRST:0 HCRST:0 R/
    S:Run
USB Status (FF82) = 0001
    HCHalted:0 HCPProcError:0 HostErr:0 ResumeDtct:0 USBErrIntr:0
    USBIntr:1
USB Interrupt Enable (FF84) = 000F
    Short Packet:1 IOC:1 Resume:1 Timeout/CRC:1
Frame Number (FF86) = 0050
FrameList BaseAddr (FF88) = 02BCE160
Start of Frame Modifier (FF8C) = 40 (12000 clocks/frame)
Port 1 Status/Control (FF90) = 0095
    Suspend:Enabled Rst:0 LowSpd:0 ResumeDtct:0 LineStat:1
    EnabChng:0 Enab:1 CStatChng:0 CStat:1
Port 2 Status/Control (FF92) = 0080
    Suspend:Enabled Rst:0 LowSpd:0 ResumeDtct:0 LineStat:0
    EnabChng:0 Enab:0 CStatChng:0 CStat:0
Legacy Support = 00003F00
    A20PTS:0 USBPIRQDen:1 USBIRQS:1 TBy64W:1 TBy64R:1 TBy60W:1
    TBy60R:1
    SMIEPTE:0 PSS:0 A20PTEn:0 USBSMIEn:0 64WEn:0 64REn:0 60WEn:0
    60REn:0
```

Here is some of the output from the -schedule command. This example shows only the first few entries; the complete USB schedule is quite long.

```
USB -sc 0

USB Transaction Schedule for Host Controller 0:
Universal Host Controller at PCI Bus 0 Device 31 Function 2
USB schedule at 827CE000

-----

Frame 0 at 827CE000
====Queue entry at 02DAF000====
Horiz Link Ptr: 02BCF3C0 (Queue:1 T:0)
Vert Link Ptr: 02DAF100 (Queue:0 T:0)
-----TD at 02DAF100-----
Next Entry: 00000000 (Vf:0 Queue:0 T:1)
SPD:1 C_ERR:3 LS:0 ISO:0 IOC:1 ActLen:800 bytes
Status (Act:1 Stalled:0 DBErr:0 Babble:0 NAK:1 CRC/TMout:0
BitErr:0)
MaxLen: 1 DataPID:0 EndPoint: 1 DevAddr: 1 PID: 69
Buffer address: 02F16E70
=====End Q=====
Frame 1 at 827CE004
====Queue entry at 02B5E000====
Horiz Link Ptr: 02BCF100 (Queue:1 T:0)
Vert Link Ptr: 02D26460 (Queue:0 T:0)
-----TD at 02D26460-----
Next Entry: 02D26380 (Vf:0 Queue:0 T:0)
SPD:1 C_ERR:3 LS:1 ISO:0 IOC:0 ActLen:800 bytes
Status (Act:1 Stalled:0 DBErr:0 Babble:0 NAK:1 CRC/TMout:0
BitErr:0)
MaxLen: 4 DataPID:1 EndPoint: 1 DevAddr: 2 PID: 69
Buffer address: 02D26470
```


VER

Type

Miscellaneous

Definition

Display the SoftICE version number.

Syntax

VER

Note: 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.

```
VER
SoftICE (R) - DriverStudio (tm) 4.3.1 (Build 1940)
Windows NT Version 5.0 - Build 2195 (Free) SP 3
```

WATCH

Type

Watch

Definition

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

The following example creates an entry in the Watch window for the variable `hInstance`.

```
WATCH hInstance
```

The following 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 lpzCmdLine. The results show the value of the pointer (0x8158D72E) and the ASCII string (currently null).

```
WATCH lpzCmdLine +char * =0x8158D72E <"">
```

Double-clicking on this line expands it to show the actual string contents.

```
lpzCmdLine -char * =0x8158D72E  
      char = 0x0
```

See Also

WW

WC

Type

Window Control

Key

Alt + F3

Definition

Toggle the Code window open or closed; and set the size of the Code window.

Syntax

```
WC [-o | -c][+ | -][window-size]
```

+ / - Optional switch to increase or decrease the window size by the decimal number referred to in *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 specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

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
```

The following example expands the twelve-line code window (set in the previous example) to eighteen lines.

```
WC +6
```

See Also

WD; WF; WL; WR; WS; WT; WW; WX

WD

Type

Window Control

Key

Alt-F2

Definition

Toggle Data window *n* open or closed, and optionally, set the size of Data window *n*.

Syntax

```
WD[.0 | .1 | .2 | .3 | (Data Window Number)][-o | -c]  
[ + | - ][window-size]
```

*.0 | .1 | .2 | .3 | (Data
Window Number)*

Optional switch to identify the Data Window Number.

-o

Forces a window open.

-c

Forces a window closed.

+ / -

Optional switch to increase or decrease the window size by the decimal number referred to in *window-size*.

window-size

Decimal number.

Use

If you do not specify the *window-size* or *data window number*, the **WD** command toggles Data window 0 (the default Data window) open or closed. If Data window 0 is closed, **WD** opens it; and if it is open, **WD** closes it.

If you *do* specify the *data window number* and *window-size*, Data window *n* is resized. If it is closed, the command opens it to the specified size.

If you specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

When a Data window is closed, the extra screen lines from the Data window are added to the Command window. When a 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 Data window 2 is closed, the following example displays the window and sets it to twelve lines. If Data window 2 is open, the example sets it to twelve lines.

```
WD.2 12
```

The following example expands the twelve-line Data window 2 (set in the previous example) to eighteen lines.

```
WD.2 +6
```

See Also

WC; WF; WL; WR; WS; WT; WW; WX

WF

Type

Window Control

Key

CTRL-F3

Definition

Display the floating point stack in either floating point or MMX format.

Syntax

```
WF [-o | -c][-d] [b | w | d | f | p | *]
```

<i>-o</i>	Forces a window open.
<i>-c</i>	Forces a window closed.
<i>-d</i>	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.
<i>b</i>	Display the floating point stack in byte hexadecimal format.
<i>w</i>	Display the floating point stack in word hexadecimal format.
<i>d</i>	Display the floating point stack in dword hexadecimal format.
<i>f</i>	Display the floating point stack in 10-byte real format.
<i>p</i>	Display the floating point stack as packed 32-bit floating point. This is the AMD 3DNow format.
<i>*</i>	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 you specify `-o`, the window is forced open. If you specify `-c`, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

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

The following example shows the use of the WF command with the `-d` option set to show the floating point stack, and the `-f` option set to display the stack in 10-byte real format.

```
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:

```
bits(63-48) bits(47-32) bits(31-16) bits(15-0)
```

See Also

WC; WD; WL; WR; WS; WT; WW; WX

WHAT

Type

System Information

Definition

Determine if a name or expression is a “known” type.

Syntax

WHAT [*name* | *expression*]

name Any symbolic name that cannot be evaluated 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* 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* parameter is something that would not generally be considered a name. 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 because 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.

Since 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'
```

WIDTH

Type

Customization

Definition

Set the number of display columns in the SoftICE window.

Syntax

WIDTH [*80-160*]

80 - 160 The number of display columns.

Use

When you are using SoftICE with the Universal Video Driver, you can use the WIDTH command can be used to set the number of display columns between 80 and 160. The default width is 80.

If you enter the WIDTH command without specifying a parameter, SoftICE displays the current setting of the window's width.

Example

The following example sets the width of the SoftICE window to 90 display columns.

```
WIDTH 90
```

The following command returns the current width setting of the SoftICE window.

```
WIDTH
```

See Also

LINES; SET

WINERROR

Type

System Information

Definition

Display header-defined mnemonics for Win32 error codes.

Syntax

WINERROR *code*

code The Win32 error code associated with a specific header-defined mnemonic.

Use

The WINERROR command displays the header-defined mnemonic associated with a specific Win32/64 error code. This command allows you to return the more intuitive mnemonic associated with any Win32 error code.

Example

The following example shows the WINERROR command returning the mnemonic for the error code 0x103:

```
WINERROR 0x103
ERROR_NO_MORE_ITEMS
```

WL

Type

Window Control Command

Definition

Toggle the Locals window open or closed; and set the size of the Locals window.

Syntax

```
WL [-o | -c][+ | -][window-size]
```

<i>-o</i>	Forces a window open.
<i>-c</i>	Forces a window closed.
<i>+ / -</i>	Optional switch to increase or decrease the window size by the decimal number referred to in <i>window-size</i> .
<i>window-size</i>	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.

If you specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

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.

Note: 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. Expandable items are indicated with a plus mark (+).

Example

If the Locals window is closed, the following example displays the window and sets it to twelve lines. If the Locals window is open, the example sets it to twelve lines.

```
WL 12
```

The following example expands the twelve-line code window (set in the previous example) to eighteen lines.

```
WL +6
```

See Also

LOCALS; TYPES; WC; WD; WF; WR; WS; WT; WW; WX

WMSG

Type

System Information

Definition

Display the names and message numbers of Windows messages.

Syntax

```
WMSG [partial-name* | wmsg-number]
```

<i>partial-name*</i>	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.
<i>msg-number</i>	Hexadecimal message number of the message. Only the message that matches the msg-number displays.

Use

The following 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.

Examples

The following example displays the names and message numbers of all Windows messages that start with *wm_get*.

```
WMSG wm_get*
000D WM_GETTEXT
000E WM_GETTEXTLENGTH
0024 WM_GETMINMAXINFO
0031 WM_GETFONT
0087 WM_GETDLGCODE
```

The following example displays the Windows message that has the specified message number, 111.

```
WMSG 111
0111 WM_Command
```


WR

Type

Window Control

Key

F2

Definition

Toggle the Register window.

Syntax

WR [**-o** | **-c**]

-o Forces a window open.

-c Forces a window closed.

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.

If you specify **-o**, the window is forced open. If you specify **-c**, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

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.

The WR command also toggles the visibility of the floating point Register window if one is open.

See Also

WC; WD; WF; WL; WS; WT; WW; WX

WS

Type

Window Control

Key

ALT-S

Definition

Toggle the call stack window open or closed, and set the size of the stack window.

Syntax

```
WS [-o | -c][+ | -][window-size]
```

-o Forces a window open.

-c Forces a window closed.

+ / - Optional switch to increase or decrease the window size by the decimal number referred to in *window-size*.

window-size The number of lines in the SoftICE window assigned to the call stack window.

Use

You can use the arrow keys to select a particular call stack element. When you select a call stack item and press Enter, SoftICE updates the Locals and Code windows to show the selected stack level. You can also click your mouse in the Stack window to set focus, single-click an item to select it, and double-click an item to update the Locals and Code windows.

If you specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

Example

The following command open the call stack window, if it is closed, and sets its size to twelve lines.

```
WS 12
```

The following example expands the twelve-line code window (set in the previous example) to eighteen lines.

```
WS +6
```

See Also

WC; WD; WF; WL; WR; WT; WW; WX

WT

Type

Window Control

Key

Alt-T

Definition

Toggle the Thread window open or closed; set the size of the Thread window.

Syntax

```
WT [-o | -c][window-size][+ | -][window-size]
```

-o Forces a window open.

-c Forces a window closed.

+ / - Optional switch to increase or decrease the window size by the decimal number referred to in *window-size*.

window-size Decimal number.

Use

If you do not specify the *window-size*, WT toggles the window open or closed. If the Thread window is closed, WT opens it; if it is open, WT closes it.

If you specify the *window-size*, the Thread window is resized. If it is closed, WT opens it to the specified size.

If you specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

The WT command displays information on threads for a given process. By using the ADDR command, you can switch to a different process context (thereby switching the viewable threads).

The information displayed for each thread includes the thread ID, KTEB, UTEB, state, process (ID #), and attributes. Information under the attributes can be any combination of the following:

<i>*</i>	Identifies the currently executing thread.
<i>NP</i>	Means that the thread's frame has been paged out (i.e., Not Present).
<i>S</i>	Means that the particular entry is selected and the Code, Stack, and Locals windows are referring to that thread.

If you wish to move the cursor to the Thread window, use the Alt-T hotkey. By navigating into the Thread window and selecting one of the threads, the Stack, Code, and Locals windows will be updated to reflect the thread that is selected. If the selected thread's frame is paged out, the Stack, Code, and Locals windows will default to the currently executing thread.

Example

If the Thread window is closed, the following example displays the window and sets it to twelve lines. If the Thread window is open, this example sets it to twelve lines.

```
WT 12
```

The following example expands the twelve-line code window (set in the previous example) to eighteen lines.

```
WT +6
```

See Also

WC; WD; WF; WL; WR; WS; WX; THREAD

WW

Type

Window Control

Key

Alt-F4

Definition

Toggle the Watch window open or closed, set the size of the Watch window.

Syntax

```
WW [-o | -c][window-size][+ | -][window-size]
```

-o Forces a window open.

-c Forces a window closed.

+ / - Optional switch to increase or decrease the window size by the decimal number referred to in *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; 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.

If you specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

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 twelve lines. If the Watch window is open, this example sets it to twelve lines.

```
WW 12
```

The following example expands the twelve-line code window (set in the previous example) to eighteen lines.

```
WW +6
```

See Also

WC; WD; WF; WL; WR; WS; WT; WX

WX

Type

Window Control

Definition

Toggle the XMM register window open or closed; set the display format of the window.

Syntax

```
WX [-o | -c] [D | Q | DQ | SF | DF | *]
```

<i>-o</i>	Optional switch to force a window open.
<i>-c</i>	Optional switch to force a window closed.
<i>D</i>	Display as 32-bit dword values.
<i>Q</i>	Display as 64-bit quadword values.
<i>DQ</i>	Display as 128-bit double quadword values.
<i>SF</i>	Signed real value.
<i>DF</i>	Double real value.
<i>*</i>	Change to next format.

Use

On computers using the Pentium III (and later) CPU, you can use the WX command to display a window that contains the value of the XMM registers, XMM0 through XMM7.

If you specify *-o*, the window is forced open. If you specify *-c*, and the window is already open, it is forced closed. These options are useful for macros, as they allow you to set the screen layout to a known state.

If you use the *D* option the values are displayed as 32-bit dwords. If you use the *Q* option the values are displayed as 64-bit quadwords. If you use the *DQ* option the values are displayed as 128-bit double quadwords. If you use the *SF* option, the register values are displayed as signed real values. If you use the *DF* option, the register values are displayed as double real values. You can use an asterisk (*) to change to the next format.

Example

The following example displays the XMM register window. The values are displayed as dword values.

WX d

See Also

WC; WD; WF; WL; WR; WS; WT; WW

X

Type

Flow Control

Key

F5

Definition

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

Type

System Information

Definition

Display active exception frames.

Syntax

XFRAME [*frame address*]

frame address Address for an exception frame.

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 in which the 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, XFRAME lists each active handler for the exception frame. For each active handler, XFRAME displays its type (try/except or try/finally), the address of its exception filter (for try/except only), and the address of the exception handler. Since exception handlers can be nested, more than one entry may be listed for each ExceptionRegistration record.

The XFRAME command displays bare addresses in its output. You can use either the STACK or WHAT commands to determine the API that installed an exception handler.

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, XFRAME displays the following information:

<i>xFrame</i>	Address of the ExceptionRegistration. This value is stack based.
<i>xHandler</i>	Address of the global exception handler which dispatches the exception to the appropriate try/except/finally filter/handler.
<i>xTable</i>	Address of the scope table used by the global exception handler to dispatch exceptions.
<i>xScope</i>	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 to display information about the exception handler frames that are currently active:

XFRAME			
xFrame	xHandler	xTable	xScope
-----	-----	-----	-----
0x45FFFD	0x60639638	0x606018B8	00
	try/except (0000) filter=0x60606F72, handler=0x60606F85		
0x45FFFA	0x5FE16890	0x5FE11210	00
	try/except (0000) filter=0x5FE125EB, handler=0x5FE125F8		
0x45FFB7	0x77F8B1BC	0x77F61370	00
	try/except (0000) filter=0x77F7DD21, handler=0x77F7DD31		

ZAP

Type

Mode Control Command

Definition

Replace an embedded INT 1 or INT 3 with a NOP instruction.

Syntax

ZAP

Use

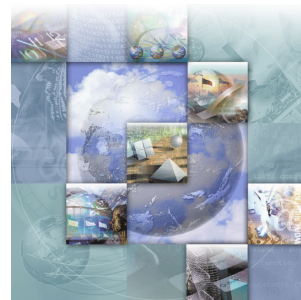
The ZAP command replaces an embedded interrupt (INT 1 or INT 3) with the appropriate number of NOP instructions. This is useful when the INT 1 or INT 3 is placed in code that is repeatedly executed, and you no longer want SoftICE to pop up. This command works only if the INT 1 or INT 3 instruction is the instruction before the current CS:EIP.

Example

The following example replaces an embedded INT 1 or INT 3 with a NOP instruction.

ZAP

Index



Symbols

. 1
? 2

A

A 3
ACTION 4
ADDR 5
Address Context 5
ALTKEY 8
ALTSCR 9
ANSWER 11
APC 13, 15
Assemble Code 3
ATTACH 15

B

BC 18
BD 19
BE 20
BH 21
BL 22
BMSG 24
BPE 26
BPINT 28
BPIO 30
BPLOAD 32
BPM 32, 34
BPR 37
BPT 37
BPX 39
BSTAT 42

C

C 44
CLASS 45
CLS 47
CODE 48
COLOR 49
CR 54
CSIP 54

D

D 56
DATA 59, 60
DETACH 60
DEVICE 60, 61
DEX 64
DIAL 68
DPC 70
DRIVER 71

E

E 74
EC 76
ERESOURCE 78
Evaluate Expressions 2
EVENT 80
EVMEM 84
EVRES 86
EXIT 86
EXP 89

F

F 91
FAULTS 92
FIBER 93
FILE 94
FKEY 96
FLASH 103
FMUTEX 104
FOBJ 105
FORMAT 107

G

G 108
GDT 109
GENINT 112

H

H 114
HBOOT 115
HEAP 116
HEAP32 119
HERE 124
HS 126
HWND 127

I

I 132
I1HERE 133
I3HERE 134
IDT 136
INTOBJ 138
IOCTL 140
IRB 141
IRP 145
IRQ 148

K

KEVENT 152
KMUTEX 153
KSEM 154

L

LDT 155
LHEAP 158
LINES 160
LOCALS 161

M

M 163
MACRO 164
MAP32 169
MAPV86 171
MOD 173
MSR 176

N

NAME 178
NAME command 178
NET 181
NTCALL 185
NTSTATUS 187

O

O 187
OBJDIR 189
OBJTAB 191
OPINFO 193

P

PACKET 197
PAGE 199
PAGEIN 202
PAUSE 203
PCI 204
PEEK 206
PHYS 207
POKE 209
Print Screen Key 210
PRN 211
PROC 212

Q

QUERY 215

R

R 220
Register
 Groups v
 Names v
RS 222

S

S 223
SERIAL 225
SET 225
SHOW 235
SRC 235
SS 236
STACK 237
SYM 241
SYMLOC 244

T

T 246
TABLE 247
TABS 249
TASK 249
THREAD 250
TIMER 253
TRACE 255
TSS 255
TYPES 257

U

U 259
USB 261

V

VER 265
VM 266

W

WATCH 266
WC 268
WD 270
WF 272
WHAT 274
WIDTH 276
WINERROR 277
WL 277
WMSG 280
WR 281
WS 282
WT 284
WW 286
WX 288

X

X 290
XFRAME 291
XG 294

Z

ZAP 294

