Visual SoftICE Command Reference

Release 1.3.0

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

Doc. 11582
May 19, 2003

Table of Contents

Preface

Visual SoftICE Commands

Preface

This reference manual explains the functionality of all Visual SoftICE commands. Commands described in this reference operate on IA-32 or IA-64 targets with the following supported operating systems:

- Windows® 2000
- Windows® XP

For each command, this reference provides information on the proper syntax, available options, expected output, examples, and related commands, as applicable.

Register Names and Groups

In order to simplify the presentation of the large number of registers available, Visual SoftICE uses the concept of Register Groups. Registers are grouped by type and function, allowing you to work with them in smaller, more logical pieces.

To view detailed information on any register group and the corresponding fields, you can use the R command with the *field*, *symbol*, and *descriptive* flags, or the Registers page and the Details status bar.

Visual SoftICE organizes the available registers into the following groups for IA-64:

State Registers	A collection of the Processor State (PSR), Instruction Pointer (IP), Current Frame Marker (CFM), and Slot registers.
General Registers	General purpose 64-bit registers are GR0 - GR127. IA-32 integer and segment registers are contained in GR8 - GR31 when executing IA-32 applications.
Local Registers	A collection of registers representing the current locals (variable length).
Predicate Registers	Single-bits used in IA-64 predication and branching are PR0 - PR63.
x86 Registers	A view of registers as seen by a 32-bit process executing on the 64-bit platform.
Floating-point Registers	Floating-point registers are FR0 - FR127. IA-32 floating-point and multi-media registers are contained in FR8 - FR31 when executing IA-32 instructions.
Floatstack Registers	Rotating floating-point registers are F0 - F127.
System Registers	A collection of critical system registers, including Task Priority, Interrupt, and Control registers.
Branch Registers	Registers used in IA-64 branching are BR0 - BR7.
Application Registers	A collection of special-purpose IA-64 and IA-32 application registers.
Perfdata Registers	Data registers for performance monitor hardware are PMD0 - PMD7.
CPUID Registers	Registers that describe processor implementation-dependent features.
Breakpoint Registers	Instruction and Data Breakpoint registers are IBR0 - IBR7 and DBR0 - DBR7.
Region Registers	Region registers are RR0 - RR7.
Protection Registers	Protection key registers are PK0 - PK15.
Translation Registers	Instruction and Data Translation registers are ITR0 - ITR7 and DTR0 - DTR7.

Visual SoftICE organizes the available registers into the following groups for IA-32:

General Registers	General purpose 32-bit registers.
Segment Registers	Segment registers.
Float Registers	Floating-point registers.
MMX Registers	Multi-media extension registers.
XMM Registers	SSE multi-media extension registers.
Debug Registers	Debug registers.
Control Registers	Control registers.

Wildcards

Visual SoftICE has the following rules regarding use of wildcard characters:

- Wildcard characters are not supported in numeric input.
- Except where noted, all commands that take "names" can also take the single character wildcard (?) and the multiple character wildcard (*). This is true for all symbols (locals, globals, function names, exports, etc.), register names and aliases, objects in commands (like DRIVER, DEVICE, etc.), and user-defined names (addresses and literals).
- Wildcard characters are not supported in macro names. You must enter a full macro name exactly as it was defined. Visual SoftICE already does partial matches against macros as though they were valid commands in the command language, so wildcard characters are meaningless in this context.

Example

`kb*fil*isr*` matches `KbFilter_IsrHook` because the first * matches zero characters, the second matches a few characters, and the last matches any number of characters (including zero) at the end. If this had been entered as `kb?fil*isr*` it would not have matched because the ? wildcard *must* match exactly one character and not zero characters.

Visual SoftICE Commands

!

Execute Kernel Debugger extension.

Syntax

`!`*`kdext.command`*

kdext Name of the Kernel Debugger extension.

command The command inside the Kernel Debugger extension to execute.

Use

Use the ! command to execute a Kernel Debugger (KD) extension command.

Example

The following example executes the **pcitree** command inside the **kdexts** KD extension:

```
SI>!kdexts.pcitree
Bus 0x0 (FDO Ext e000000086db6dd0)
  0800 123d8086 (d=0,  f=0) devext e000000086f75690 Base System Device/Interrupt Controller
  0700 2000131f (d=1,  f=1) devext e000000086f75350 Simple Serial Communications Controller/Serial Port
  0604 00241011 (d=2,  f=2) devext e000000086f75010 Bridge/PCI to PCI
...
Total PCI Root busses processed = 4
```

$

Pass a program name and command line arguments to the master side operating system to execute.

Syntax

```
$ program-name [cmd-line]
```

program-name	Name of the program to execute.
cmd-line	Command line arguments for the specified program.

Use

Use the $ command to have the master side operating system execute a specified program, including any command line arguments you want to pass to the program at execution.

Example

The following example executes Windows Notepad and has it open `myfile.txt`:

```
SI>$ notepad c:\myfile.txt
```

See Also

EXEC

.

Disassemble at the current instruction entered alone in a Command (CMD) page, or interpreted as an address in any other syntax.

Syntax

.

Use

When in the CMD page, the . (Dot) command disassembles at the current Instruction Pointer (IP). When in any other page, the . (Dot) command is interpreted as an address and used as applicable for that page.

Example

The following example shows disassembly of code using the . (Dot) command with symbols loaded:

```
SI>.
·································ntoskrnl!.KeInsertQueueDpc+10·······················
* 0xe000000083076390.s0   adds r43 = 0x0, gp
0xe000000083076390.s1   movl r40 = 0xe0000000ffff0b18 ;;
0xe0000000830763a0.s0   addl r44 = 0xf, r0
0xe0000000830763a0.s1   adds r35 = 0x2, r32
0xe0000000830763a0.s2   adds r45 = 0x10, sp ;;
0xe0000000830763b0.s0   ld8.nta r3 = [sp]
0xe0000000830763b0.s1   nop.f 0x0
0xe0000000830763b0.s2   br.call.sptk.many rp = $+0x1df0 // ( ntoskrnl!.KeRaiseIrql ) ;;
```

?

Evaluate an expression.

Syntax

```
? expression
```

expression	The expression to evaluate.

Use

To evaluate an expression, enter the ? command followed by the expression to evaluate. Visual SoftICE displays the result in decimal (signed decimal only if the value is less than 0), hexadecimal, ASCII, and binary.

You can explicitly evaluate the decimal or hexadecimal values regardless of the current radix setting by using `dec()` or `hex()` with the number in the parenthesis. The Expression Evaluator returns the corresponding value.

Example

The following command displays the hexadecimal, decimal, ASCII, and binary representations of the value of the expression `10h*4h+3h`:

```
SI>? 10*4+3
0000000000000043 (67) "C" 0100 0011
```

The following command explicitly evaluates three different expressions using `dec()` and `hex()`:

```
SI>? dec(111)
0000006f (111)
SI>? hex(111)
00000111 (273)
SI>? dec(111) + hex(111)
00000180 (384)
```

See Also

EVAL, SET EE_EVAL_ORDER, SET EE_IMPL_DEREF, SET RADIX

@

Load and execute a script file on the target.

Syntax

```
@ [file-name]
```

file-name The file name of the script to load and execute.

Use

Use the @ command to load and execute a script file on the target. Script files are any ASCII text files containing at least one Visual SoftICE command per line, or separated by semi-colons on the same line.

Example

The following example loads and executes `MyScript.txt` on the target:

```
SI>@ MyScript.txt
```

See Also

SAVE, SCRIPT, SET LOG, SET SCRIPTECHO, SET SCRIPTPATH, SET SCRIPTSTOPONERROR, SLEEP

ADDR

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 Visual SoftICE to view process-specific data and code. Using ADDR with no parameters displays a list of all address contexts.

If you specify a parameter, Visual SoftICE switches to the address context belonging to the process with that name, identifier, or process control block address. Switching the context causes on-demand symbol loading to occur.

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

Example

The following example displays a list of all address contexts:

```
SI>ADDR
PageDir           KPEB              Pid       Name
-----------------------------------------------------------
00000000035d6461  e000000081a6fa40  0         Idle
00000000149e0461  e0000000865a7c10  188       fib64_2.exe
...
000000000fbe4461  e00000008660ba90  1fc       svchost.exe
```

The following example switches address contexts to `fib64_2.exe` using its PID from the list:

```
SI>ADDR 188
```

The following example shows how the symbol tables are affected by switching address contexts. The initial address context is the idle process. Notice how the new image for the new context is automatically loaded:

```
SI>TABLE
Name                 Version  Type    Gbl Exp Status
----------------------------------------------------
__USERNAMES__*       00000001 User     Y   N  OK
*ntoskrnl.exe         3b7de38f Symbol   Y   N  Matching PDB file C:\Symbols\exe\ntoskrnl.pdb.
SICORE.SYS           3ce0df22 Symbol   Y   N  Can't find PDB symbol file.
SI>ADDR winlogon.exe
SI>TABLE
Name                 Version  Type    Gbl Exp Status
----------------------------------------------------
__USERNAMES__*       00000001 User     Y   N  OK
*ntoskrnl.exe         3b7de38f Symbol   Y   N  Matching PDB file C:\Symbols\exe\ntoskrnl.pdb.
SICORE.SYS           3ce0df22 Symbol   Y   N  Can't find PDB symbol file.
winlogon.exe         3b7d8cc8 Symbol   N   N  Matching PDB file C:\Symbols\exe\winlogon.pdb.
SI>ADDR explorer.exe
SI>TABLE
Name                 Version  Type    Gbl Exp Status
----------------------------------------------------
__USERNAMES__*       00000001 User     Y   N  OK
Explorer.EXE         3b7de06e Symbol   N   N  Matching PDB file C:\Symbols\exe\Explorer.pdb.
*ntoskrnl.exe         3b7de38f Symbol   Y   N  Matching PDB file D:\Symbols\exe\ntoskrnl.pdb.
SICORE.SYS           3ce0df22 Symbol   Y   N  Can't find PDB symbol file.
```

See Also

IMAGE, PROCESS, TABLE

QUERY (ADDRESSMAP)

Display the virtual address map of a process.

Syntax

```
QUERY [[-a] address] | [process-type]
```

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

Use

The QUERY command displays a map of 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:

Context Address context.

Address Range Start and end address of the linear range.

Flags Flags from the node structure.

MMCI Pointer to the memory management structure.

PTE Structure that contains the ProtoPTEs for the address range.

Name Additional information about the range. This includes the following:

- Memory mapped files will show the name of the mapped file.
- Executable modules will show the file name of the DLL or EXE.
- Stacks will be displayed as (thread ID).
- Thread information blocks will be displayed as TIB (thread ID).
- Any address that the WHAT command can identify might also appear.

Example

The following example uses the QUERY command to map a specific linear address for Windows NT.

```
SI>QUERY -a 77f50000
Count: 14
Context        Address Range      Flags     MMCI      PTE       Name
---------------------------------------------------------------------------
System         77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
smss.exe       77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
csrss.exe      77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
winlogon.exe   77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
services.exe   77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
lsass.exe      77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
spoolsv.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
explorer.exe   77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
siservice.exe  77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
logon.scr      77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
```

The following example uses the QUERY command to list the address map of the explorer process for Windows NT.

```
SI>QUERY explorer
Address Range      Flags     MMCI      PTE       Name
--------------------------------------------------------------------
00010000-00010000  c4000001
00020000-00020000  c4000001
00030000-0006f000  8400000f
00070000-00070000  01400000  809e5540  e10e5150
00080000-0017f000  840000b1                      Process Heap
...
7ffdc000-7ffdc000  c6400001                      Tib:338
7ffde000-7ffde000  c6400001                      Tib:13c
7ffdf000-7ffdf000  c6400001                      UPEB (20c)
```

ADDSYM

Add persistent symbols from an image or .pdb file.

Syntax

```
ADDSYM [-v] file-name
```

-v	Verbose mode.
file-name	The name of the image or .pdb file.

Use

Use the ADDSYM command to add persistent symbols from an image or .pdb file. Using ADDSYM with the -*v* option will generate verbose information about the symbol engine's search for the image or symbol file specified.

Symbols are normally loaded on-demand, however ADDSYM can be used to pre-load or maintain symbols as needed according to your debugging preferences. These symbols stay loaded until you specifically remove them, or Visual SoftICE exits.

On-demand symbol loading differs from the previous versions of SoftICE, which required you to always pre-load symbols into memory. Setting deferred (virtual) breakpoints is not possible with Visual SoftICE loading symbols on-demand. Deferred breakpoints are set before the image of interest has been loaded by the operating system, and in order to set them, you must have symbols or exports available on the master within your search path(s).

Use the ADDSYM command to set deferred breakpoints by informing the symbol engine of data you want to persistently load before you issue any Set Breakpoint commands.

Example

The following example adds symbols from a .pdb file in verbose mode:

```
SI>ADDSYM -v mypdb.pdb
```

See Also

DELSYM, FILE, GETEXP, LOAD, RELOAD, SET SYMPATH, SET SYMSRVSEARCH, SET SYMTABLEAUTOLOAD, TABLE, UNLOAD

APC

Display Asynchronous Procedure Calls.

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, Visual SoftICE lists all asynchronous procedure calls queued for delivery in the currently running thread. Or you can instruct Visual SoftICE to walk through a specified thread or process.

Example

The following command displays information about an asynchronous procedure call:

```
SI>APC 1c8
------------------------------------------------------
Address          : 81b107f8
Thread           : 81d0d8b8
KernelRoutine    : ntoskrnl!IopCompleteRequest (804e90e4)
RundownRoutine   : ntoskrnl!IopAbortRequest (80557946)
NormalRoutine    : 00000000
NormalContext    : 00000000
SystemArgument1  : 81d00748
SystemArgument2  : 00000000
ApcStateIndex    : 0
ApcMode          : 0
Inserted         : yes
```

See Also

DPC

ARBITER

Display a list of arbiters for different types of resources.

Syntax

```
ARBITER [-p | -i | -m | -b | -d]
```

-p	Display all port arbiters.
-i	Display all interrupt arbiters.
-m	Display all memory arbiters.
-b	Display all bus arbiters.
-d	Display all DMA arbiters.

Use

The ARBITER command displays a list of arbiters for different types of resources. If you use the ARBITER command without any flags set, it dumps a list of all arbiters. Use the flags to select a sub-set of arbiters to display.

Example

The following example shows the ARBITER command displaying a list of all interrupt arbiters.

```
SI>ARBITER -i
ARBITER
----------------------------------------------------
Address      : 80543960 (ntoskrnl!IopRootIrqArbiter)
Type         : 2; Interrupt
Name         : RootIRQ
Event        : 80f723a0
DeviceObject : 00000000
RefCount     : 0
RANGE
Starting Address  Ending Address  Attributes  Flags      Owner     Owner
-------------------------------------------------------------------------------
0                 0               1           0;         80f71a48  PnpManager
1                 1               1           0;         80f71a48  PnpManager
2                 2               1           0;         80f71a48  PnpManager
3                 3               1           0;         80f71a48  PnpManager
...
2f                2f              1           0;         80f71a48  PnpManager
32                32              1           0;         80f71a48  PnpManager
39                39              0           1; SHARED  80f6dba0  ACPI
ARBITER
------------------------------------------
Address      : f91b4a00 (ACPI!AcpiArbiter)
Type         : 2; Interrupt
Name         : ACPI_IRQ
Event        : 80e162c8
DeviceObject : 80f6b420
RefCount     : 0
RANGE
Starting Address  Ending Address  Attributes  Flags      Owner     Owner
-------------------------------------------------------------------------------
0                 0               1           0;         80f54ab8  ACPI
1                 1               0           0;         80f54888  i8042prt
3                 3               1           0;         80f71a48  PnpManager
4                 4               0           0;         80f54658  SISERIAL
...
d                 d               1           0;         80f54f18  ACPI
e                 e               0           0;         80e11728  atapi
f                 f               0           0;         80f52bf0  atapi
```

See Also

DEVICE, DEVNODE, DRIVER

ASSEMBLE

Assemble instructions.

Syntax

```
ASSEMBLE [address]
```

```
A [address]
```

address	Address at which you want to assemble instructions.

Use

The ASSEMBLE command assembles source code at the specified address.

If you do not specify the address, the ASSEMBLE command assembles the source code at the current Instruction Pointer (IP).

The ASSEMBLE command switches the command prompt to assembly mode, where only assembly instructions and some specific commands like HELP are valid. To exit assembly mode, you must enter a `<RETURN>` on an empty line.

Example

The following example assembles instructions beginning at the address offset `my_function+120`:

```
SI>ASSEMBLE my_function+120
```

The following command assembles instructions at the current IP:

```
SI>ASSEMBLE
```

BC

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:

```
SI>BC *
```

To clear breakpoints 1 and 5, use the following command:

```
SI>BC 1 5
```

See Also

BD, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BD

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.

```
SI>BD 1 3
```

See Also

BC, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BE

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).

Example

To enable breakpoint 3, use the following command:

```
SI>BE 3
```

See Also

BC, BD, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, BPSTAT, SET GLOBALBREAK

BL

List all breakpoints.

Syntax

```
BL
```

Use

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

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

The BL command has no parameters.

Example

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

```
SI>BL
2 breakpoints
  (0)             BPX (EXECUTE-INSTR)  (0000000000403440.s0) testexe!main
(1)             BPX (EXECUTE-INSTR)  (e0000000830451c0.s0) ntoskrnl!.IofCallDriver
```

See Also

BC, BD, BE, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BMSG

Set a breakpoint on one or more Windows messages.

Syntax

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

-l	Enable logging for this breakpoint.
window-handle	HWND value returned from CreateWindow or CreateWindowEX.
begin-msg	Single Windows message or lower message number in a range of Windows messages. If you do not specify a range with an *end-msg*, only the *begin-msg* will cause a break.

Note: For both *begin-msg* and *end-msg*, the message numbers can be specified either in hexadecimal or by using the actual ASCII names of the messages, for example, WM_QUIT.

end-msg	Higher message number in a range of Windows messages.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of Visual SoftICE commands to be executed when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered.

Use

The BMSG command is used to set breakpoints on a window 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.

When Visual SoftICE does stop on a BMSG breakpoint, the instruction pointer is set to the first instruction of the message handling procedure. Each time Visual 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 might not overlap.

Example

This command sets a breakpoint on the message handler for the Window that has the handle 9BC. The breakpoint triggers and Visual SoftICE stops 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.

```
SI>BMSG 9BC wm_mousefirst wm_mouselast
```

The next command places a breakpoint on the message handler for the Window with the handle F4C. The message range on which the breakpoint triggers includes any message with a type value less than or equal to WM_CREATE.

```
SI>BMSG f4c 0 wm_create
```

See Also

BC, BD, BE, BL, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BPINT

Set a breakpoint on an interrupt.

Syntax

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

-l	Enable logging for this breakpoint.
int-number	Interrupt number from 0 to FFh.
service-address	Specific address to differentiate between OS intobj service routines.
IF expression	Conditional expression: the expression must evaluate to TRUE (non-zero) for the breakpoint to trigger.
DO command	Breakpoint action: A series of Visual SoftICE commands that execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered.

Use

For x86 processors, use the BPINT command to stop Visual SoftICE whenever a specified processor exception, hardware interrupt, or software interrupt occurs. For IA-64 processors, use the BPINT command to stop Visual SoftICE whenever a specified hardware 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 Visual SoftICE commands that execute any time the interrupt breakpoint triggers.

For breakpoints that trigger for hardware interrupts or processor exceptions, the instruction pointer at the time Visual SoftICE stops points to the first instruction of the interrupt or exception handler routine.

The optional *service-address* parameter allows you to set a breakpoint on a shared interrupt. By passing a specific address to BPINT, it will differentiate between OS intobj service routines. If you do not specifiy a service-address, BPINT sets a breakpoint on each routine it finds that matches the vector.

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

Example

The following example results in a Windows NT system call breakpoint (software interrupt 2Eh) being triggered if the thread making the 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.

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

See Also

BC, BD, BE, BL, BMSG, BPIO, BPLOAD, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BPIO

Set a breakpoint on an I/O port access.

Syntax

```
BPIO [-l] port [verb] [IF expression] [DO
"command1;command2;..."]
```

-l	Enable logging for this breakpoint.
port	Byte or word value.

verb	**Value**	**Description**
	R	Reads (IN)
	W	Writes (OUT)
	RW	Reads and Writes

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

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered.

Use

Use the BPIO instruction to have Visual SoftICE stop whenever a specified I/O port is accessed in the indicated manner. On x86 systems, when a BPIO breakpoint triggers, the instruction pointer points to the instruction following the IN or OUT instruction that caused the breakpoint. On IA-64 systems, when a BPIO breakpoint triggers, the instruction pointer points to the actual 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 and IA64 class machines, therefore, I/O breakpoints are limited to the number of available debug registers on that hardware.

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. A drawback of the debug register method for trapping port I/O is that it does not trap emulated I/O such as I/O performed from a DOS machine.

Example

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

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

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

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

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

See Also

BC, BD, BE, BL, BMSG, BPINT, BPLOAD, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BPLOAD

Set a breakpoint on an image load.

Syntax

```
BPLOAD [-once] image-name [DO "command1;command2;..."]
```

-once	Execute the breakpoint only once.
image-name	Name of the image file on which to set the breakpoint.
DO command	Breakpoint action: A series of Visual SoftICE commands that execute when the breakpoint triggers.

Use

Use the BPLOAD command to stop Visual SoftICE whenever a specified image file loads. You can use the DO option to specify Visual SoftICE commands that execute any time the breakpoint triggers. You do not specify a path with the image name. BPLOAD cannot accept wildcards of any kind, however if you do not specify an extension the breakpoint stops on the next executable image loaded that matches the filename, regardless of the extension type (i.e., COM, DLL, EXE, or SYS).

Visual SoftICE supports a single break-on-load breakpoint for early stopping when the operating system loads a named image. This can be very useful for capturing what is going on in the load cycle of an executable. Once stopped in the image of interest, the symbols for that image should be automatically loaded, if they can be found through the search hierarchy.

BPLOAD breakpoints on an operating system driver will stop at the very beginning of DriverEntry.

Note: BPLOAD breakpoints cannot be disabled. If you want to emulate the behavior of disabling BPLOAD breakpoints, remove them and reapply them later.

Example

The following example sets a breakpoint that will stop the next time the UXTHEME.DLL image is loaded.

```
SI>BPLOAD UXTHEME.DLL
```

See Also

BC, BD, BE, BL, BMSG, BPINT, BPIO, BPM, BPR, BPX, BSTAT, SET GLOBALBREAK

BPM

Set a breakpoint on memory access or execution.

Syntax

```
BPM[size] [-l] address [verb] [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. The size is also important if you specify the optional qualifier.

Value	**Description**
B	Byte (default)
W	Word
D	Double Word
Q	Quad Word

-l Enable logging for this breakpoint.

address Address on which the breakpoint is to be set.

verb

Value	**Description**
R	Read
W	Write
RW	Read and Write (default)
X or E	Execute

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

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

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered.

Use

Use BPM breakpoints to have Visual SoftICE stop 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 Visual SoftICE commands that execute each time the breakpoint is hit. If you use BPM without specifying a size, Visual SoftICE assumes a byte size and executes BPMB.

Note: On IA-64 platforms, BPM execution class breakpoints are only allowed on a per-bundle basis. You may have no more than one BPM execution breakpoint per bundle. Visual SoftICE always sets the breakpoint on slot 0.

Visual SoftICE uses the first available debug register on the target, starting with the last sequential debug register number and working backwards. For example, if you have 4 debug registers, Visual SoftICE starts with debug register 3 and works backward to debug register 0 until it finds an available one, and it uses that register.

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

For all the verb types *except* X and E, Visual SoftICE stops after the instruction that causes the breakpoint to trigger has executed, and the Instruction Pointer points to the instruction in the code stream following the trapped instruction. For the X and E verbs, Visual SoftICE stops before the instruction causing the breakpoint to trigger has executed, and the Instruction Pointer 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.

Example

The following example defines a breakpoint on memory word access to the address pointed at by `es:di+if`. The first time that 10 hex is written to that location, the breakpoint triggers.

```
SI>BPMW es:di+1f w if (*(es:di+1f)==0x10)
Breakpoint [ 0 ] set.
```

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

```
SI>BPMB ds:80150000 w if (byte(*ds:80150000)>5)
Breakpoint [ 0 ] set.
```

See Also

BC, BD, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPR, BPX, BSTAT, SET GLOBALBREAK

BPR

Set a breakpoint on a memory range.

Note: Only available on IA64.

Syntax

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

```
BPR [-l] start-address L length [verb] [IF expression] [DO
"command1;command2;..."]
```

-l	Enable logging for this breakpoint.
start-address	Beginning of memory range.
end-address	Ending of memory range.
L length	Length in bytes.

verb	**Value**	**Description**
	R	Read
	W	Write
	RW	Read and Write
	X or E	Execute

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

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered.

Use

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

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

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

The range between the start address and end address is limited to the image address space where you set the breakpoint. If you set the breakpoint range to exceed the scope of the image address space, Visual SoftICE truncates the range to the end of the image address space.

When a range breakpoint is triggered, Visual SoftICE stops the target and the current Instruction Pointer (IP) points to the instruction that caused the breakpoint.

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

Example

The following example defines a breakpoint on a memory range. The breakpoint occurs if there are any writes to the memory between addresses 0x402000 and 0x412500.

```
SI>BPR 0x402000 0x412500 w
```

See Also

BC, BD, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPX, BSTAT, SET GLOBALBREAK

BPX

Set a breakpoint on execution.

Syntax

```
BPX [-l] [address] [IF expression] [DO "command1;command2;..."]
```

-l Enable logging for this breakpoint.

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 Visual SoftICE commands that execute when the breakpoint triggers.

Note: You can combine breakpoint count functions (BPCOUNT, BPMISS, BPTOTAL, and BPINDEX) with conditional expressions to monitor and control breakpoints based on the number of times a particular breakpoint has or has not triggered.

Use

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

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.

The BPX command accepts any valid symbol as an address parameter.

Example

The following example sets a breakpoint on a symbol:

```
SI>BPX fib64_2!fib_func
```

The following example sets a breakpoint on an address:

```
SI>BPX 434860
```

See Also

BC, BD, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BSTAT, SET GLOBALBREAK

BSTAT

Display statistics for one or all breakpoints.

Syntax

```
BSTAT [n]
```

n	Breakpoint index number.

Use

Use BSTAT to display statistics on breakpoint hits, misses, and whether breakpoints stopped Visual SoftICE or were logged.

Using BSTAT without any arguments returns statistics on all current breakpoints.

Because 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 reported in the Error column of the BSTAT output.

Note: BSTAT does not report statistics for BPLOAD breakpoints.

Output

For each breakpoint, Visual SoftICE displays the following information.

ID Breakpoint index, and if the breakpoint is disabled, an * (asterisk).

Totals Category:

Triggered Total number of times Visual SoftICE has evaluated the breakpoint.

Total Hits Total number of times the breakpoint has evaluated TRUE.

Total Miss 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 Category:

Hits Current number of times the breakpoint has evaluated TRUE, but did not stop because the count had not expired. (Refer to expression macro BPCOUNT.)

Miss Current number of times the breakpoint has evaluated FALSE or the breakpoint count has not expired.

Example

The following is an example using the BSTAT command for breakpoint #1:

```
SI>BSTAT 1
Id        Triggered  Total Hits  Total Miss  Errors    Hits      Miss
----------------------------------------------------------------------
1         3          3           0           0         3         0
```

See Also

BC, BD, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, SET GLOBALBREAK

C

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

Use the C command to compare two memory blocks. The memory block specified by *start-address* and *length* is compared to the memory block specified by *start-address-2*.

When a byte from the first data block does not match a byte from the second data block, Visual SoftICE displays both bytes and their addresses.

Example

The following example compares two data blocks at the addresses provided.

```
SI>C e0000165e57576a0.s0 L 8 e0000165e57576a1
0c b0
b0 00
00 @
@ 00
```

CLASS

Display information on Windows classes.

Syntax

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

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

Use

The architecture of class information under Windows NT/XP is similar to that of Windows 9x in that class information is process specific and the operating system creates different lists for global and private classes. Beyond this, the two operating systems have significant differences in how super-classing a registered window class is implemented.

Under Windows NT, registered window classes are considered templates that describe the base characteristics and functionality of a window (similar to the C++ notion of an abstract class). When a window is created, its class template is copied (its structure). This information is considered instance data (an instance of the class), and is stored with the other windows instance data. Any changes to the instanced class do not affect the original template. This concept is further extended when various members of the windows instanced class structure are modified. When this occurs, the parent class is referenced again, and the new instance points to the original instance. Registered classes act as templates from which copies of a particular class can be created; in effect this is a form of 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 an image of the same name. To search by image when there is a name conflict, use the image handle (base address or image database selector). Also, image names are *always* context-sensitive. If the image is not loaded in the current context (or the CSRSS context), the CLASS command interprets the image name as a class name instead.

Output

For each class, the following information is shown:

Address Offset of a data structure within USER. Refers to windows of this class.

Class Name Name that was passed when the class was registered. If no name was passed, the atom displays.

Module Name Image that has registered this window class.

WindowProc Address of the window procedure for this window class.

Styles Bitmask of flags specified when the class was registered.

Example

The following example uses the CLASS command to display all the classes registered by the explorer process.

```
SI>ADDR explorer
SI>CLASS
Count: 93
Address   Class Name                 Module Name   WindowProc
---------------------------------------------------------------
bc6432c0  PrintTray_Notify_WndClass  Explorer.EXE  bf86248a
bc6431e0  PrintUI_QueueCreate        printui.dll   74b8170e
bc643140  PrintUI_PrinterQueue       printui.dll   74ba008d
bc6411e8  Connections Tray           NETSHELL.dll  75cf1680
...
```

Note: If a symbol is not available for the window procedure, a hexadecimal address displays.

See Also

ADDR

CLOSE

Close the connection to the target machine.

Syntax

```
CLOSE
```

Use

Use the CLOSE command to close the connection to the target machine.

Example

The following example closes the connection to the target machine:

```
SI>CLOSE
```

See Also

CONNECT, DISCONNECT, NETFIND, OPEN, WCONNECT

CLS

Clear the Command window.

Syntax

```
CLS
```

Use

The CLS command clears the Command window.

Example

The following example clears the Command window:

```
SI>CLS
```

CONNECT

Connect to a target machine.

Syntax

```
CONNECT com# [-baud #] [-rt #] [-r #]
CONNECT nnn.nnn.nnn.nnn [password] [-rt #] [-r #]
CONNECT hostname [password] [-rt #] [-r #]
```

com#	Specifies the COM port for serial connections. COM1 through COM4 are valid. When connecting through a serial connection you can also specify the baud rate, retry timeout, and retry count.
IP	The IP address of the target (nnn.nnn.nnn.nnn). When connecting through an IP address you can also specify a password, retry timeout, and retry count.
hostname	The host name of the target. DNS matches the host name to its IP address and connects through the IP address. You can specify a partial host name and Visual SoftICE will match it to the complete host name if it can. You can also specify a password, retry timeout, and retry count.
-baud #	Specifies the baud rate for serial connections. Default is 115200.
password	Specifies a password to connect via IP address if the target is password protected.
-rt #	Specifies the retry timeout value. Default is 20ms.
-r #	Specifies the retry count value. Default is 5.

Note: Numeric values (baud, retry timeout, retries are entered in decimal).

Use

Use the CONNECT command to connect to the target machine. You can connect to the target machine either by serial connection or by IP address. When connecting by IP address, you can supply either the IP address, or enough of a recognized host name for Visual SoftICE to complete a DNS lookup.

Example

The following example opens a serial connection to the target:

```
SI>CONNECT com2 -rt50 -r10
Connected to:
        Name           : SPILLANE
 Processor      : IA32(x86)-Pentium III
 Stepping       : 0
 Processor Count: 1
 Operating Sys. : Windows NT/XP Ver. 5.1 Build 2600
 Target Agent   : Connected (Active)
```

The following example opens a connection to the target using its IP address:

```
SI>CONNECT 255.255.255.0
Connected to:
        Name           : mytarget-IA64
 Processor      : IA64-Itanium
 Stepping       : 0
 Processor Count: 2
 Operating Sys. : Windows XP-64 Ver. 5.1 Build 2600
 Target Agent   : Connected (Active)
```

The following example opens a connection to the target using its hostname:

```
SI>CONNECT mytarget-IA64
Connected to:
        Name           : mytarget-IA64
 Processor      : IA64-Itanium
 Stepping       : 0
 Processor Count: 2
 Operating Sys. : Windows XP-64 Ver. 5.1 Build 2600
 Target Agent   : Connected (Active)
```

See Also

CLOSE, DISCONNECT, NETFIND, OPEN, WCONNECT

CPU

Display the processor details.

Syntax

```
CPU [-i] [processor-number]
```

-i	Display the I/O Advanced Program Interrupt Controller (APIC).
processor-number	Designate the CPU number.

Use

The CPU command shows the processor information for the target.

If the target contains a multi-processor motherboard that uses an I/O APIC as an interrupt controller, the CPU command displays the CPU data and the I/O APIC information.

Examples

The following example lists the sample output from the CPU command under Windows XP on an IA64 target:

```
SI>CPU
CPU
------------------------------------
VendorString   : GenuineIntel
Class          : IA64
Model          : Itanium
Stepping       : 0
FeatureBits    : 0;
MHZ            : 2dd
Ip Address     : 0000000000403440.s0
CurrentThread  : e000000086ba0040
DpcTime        : e
InterruptTime  : 493
KernlTime      : 43e16
UserTime       : 408
InterruptCount : 52502
LOCAL APIC
--------------------------------------
Local ID                    : 0
Task Priority               : 10000
PendingInterrupts0          : 0
PendingInterrupts1          : 0
PendingInterrupts2          : 0
PendingInterrupts3          : 0
IntervalTimerVector         : d0
PerformanceMonitorVector    : f0
CorrectedMachineCheckVector : 30
LocalRedirection0           : 10000
LocalRedirection1           : 10000
```

The following example lists the sample output from the CPU command under Windows NT on an IA32 system that uses an I/O APIC:

```
SI>CPU
CPU
-------------------------------------------------------------
VendorString   : GenuineIntel
Class          : IA32(x86)
Model          : Pentium II
Stepping       : 0
FeatureBits    : 1fff;
FPU|VME|DE|PSE|TSC|MSR|PAE|MCE|CX8|APIC|SEP|MTRR
MHZ            : 14c
Ip Address     : 01003134
CurrentThread  : 80947560
DpcTime        : e
InterruptTime  : 76
KernlTime      : 2df10
UserTime       : 11e7
InterruptCount : 606a6
LOCAL APIC
--------------------------------
Local ID              : 0
Version               : 40011
Task Priority         : ff
Arbitration Priority  : ff
Processor Priority    : ff
Logical Destination   : 1000000
Spurious Vector       : 11f
Interrupt Command     : 4003d
LVT (Timer)           : 300fd
LVT (Lint0)           : 1001f
LVT (Lint1)           : 84ff
LVT (Error)           : e3
Initial Timer Count   : 3f66780
Current Timer Count   : 1d82710
Timer Divide Register : b
```

See Also

R

D

Display memory.

Syntax

```
D[size] [-p] [-e] [[address] L [count]]
```

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

-p	Indicates Physical memory address. The default is a virtual memory address.
-e	Indicates *count* parameter is not in bytes, but in element size (byte, word, DWORD, etc.)
address	Starting address of the memory to display.
L count	Displays *count* number of bytes, or elements, in the Command window.

Use

The D command displays the values stored at the specified address.

Visual SoftICE displays the memory contents in the format you specify in the *size* parameter. If you do not specify a size, Visual SoftICE uses the last size specified. For all formats, the ASCII representation is displayed.

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

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 be displayed for real formats:

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

Example

The following example displays FFh bytes of memory at address 80d9c5b0.

```
SI>DB 80d9c5b0 L ff
80d9c5b0:  3f 01 00 00 09 00 00 00 58 24 4e e1 00 00 00 00     ?.......X$N.....
...
80d9c680:  70 c6 d9 80 00 00 01 00 00 00 00 00 00 00 00 00     p...............
80d9c690:  00 c6 d9 80 00 00 00 00 00 00 00 00 00 00 00 00     ................
80d9c6a0:  00 00 00 00 00 00 00 00 00 c6 d9 80 00 00 00     ...............
```

DELSYM

Delete a persistent reference to symbols from a previous ADDSYM.

Syntax

```
DELSYM image-name
```

image-name	The name of the image or .pdb file from which to delete the persistent reference to symbols.

Use

Use the DELSYM command to delete a persistent reference to symbols from a previous ADDSYM.

Example

The following example deletes the persistent reference to symbols from an image file:

```
SI>DELSYM myfile.exe
```

The following example deletes the persistent reference to symbols from a .pdb file:

```
SI>DELSYM mypdb.pdb
```

See Also

ADDSYM, FILE, GETEXP, LOAD, RELOAD, SET SYMSRVSEARCH, UNLOAD

DEVICE

Display information on Windows NT/XP devices.

Syntax

```
DEVICE [-s] [device-name | pdevice-object]
```

-s	Dump the device stack containing the device.
device-name	Object directory name of the device.
pdevice-object	Object address of the device.

Use

The DEVICE command displays information on Windows NT/XP device objects. If the DEVICE command is entered without parameters, summary information displays for all device objects found in the operating system Device Object Collection 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`.

If you use the -s flag, Visual SoftICE dumps the device stack containing the device.

Output

The following fields are shown as summary information:

RefCount Device object’s reference count.

Address Address of a DEVICE_OBJECT structure.

DrvObj Pointer to the driver object that owns the device object.

NextDev Pointer to the next device object on the linked list of device objects that were created by the same driver.

AttachDev 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.

AttachedTo Address of the device to which this device is attached.

DevExt Pointer to device driver-defined device object extension data structure.

DevName Name of the device, if it has one.

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

DevFlags Definition of the device object’s attributes such as whether I/O performed on the device is buffered or not.

CurrentIrp Address of an IRP currently active in a device queue.

DevChar Set when a driver calls IoCreateDevice with one of the following values: FILE_REMOVEABLE_MEDIA, FILE_READ_ONLY_DEVICE, FILE_FLOPPY_DISKETTE, FILE_WRITE_ONCE_MEDIA, FILE_DEVICE_SECURE_OPEN.

VPB Pointer to the device’s associated volume parameter block.

DevType Set when a driver calls IoCreateDevice as appropriate for the type of underlying device. A driver writer can define a new FILE_DEVICE_*XXXXX*, where *XXXXX* is a value in the customer range of 32768 to 65535, if none of the system-defined values describes the type of the new device.

DevStkSize Specifies the minimum number of stack locations in IRPs to be sent to this driver.

DevQueue Device queue object for system-managed IRP queueing.

DevDpc Embedded DPC for use with *IoInitializeDpcRequest* and *IoRequestDpc*.

ActiveThreads Exclusively used by the file system to keep track of the number of FSP threads currently using the device.

SecDesc Data structure used to hold per-object security information.

DevLock Synchronization-type event object allocated by the I/O Manager.

Example

The following example uses the DEVICE command with the UDP device object's name.

```
SI>DEVICE UDP
---------------------------------------------------
Address       : e000000086bfabc0
RefCount      : 9
DevFlags      : 50; DO_DIRECT_IO|DO_DEVICE_HAS_NAME
DrvObj        : e000000086c05e70
...
DevName       : Udp
```

The following example uses the DEVICE command with the -s flag for the cdrom1 device.

```
SI>DEVICE -s cdrom1
DEVICE STACK
RefCount  Address   DrvObj    NextDev   AttachDev  AttachedTo  DevExt    DevName
--------------------------------------------------------------------------------------------
0         80d0f020  80d21a40  80d109f8  00000000   80d0e3f8    80d0f0d8
0         80d0e3f8  80d117f8  80d10030  80d0f020   80d0e8c8    80d0e4b0  CdRom1
0         80d0e8c8  80d217a8  00000000  80d0e3f8   80e3dc48    80d0e980
0         80e3dc48  80e3f750  80e5eab8  80d0e8c8   00000000    80e3dd00  IdeDeviceP1T1L0-17
```

See Also

ARBITER, DEVNODE, DRIVER

DEVNODE

Display information about device nodes.

Syntax

```
DEVNODE [-c | -rr | -rt | -x | -a | -p] address | service-name
```

-c	Dump the list of children for a device node at a specific address.
-rr	Dump the raw resource list for a device node at a specific address.
-rt	Dump the translated resource list for a device node at a specific address.
-x	Dump extended information for a device node at a specific address.
-a	Dump the list of arbiters for a device node at a specific address.
-p	Dump the parent tree for the device node.
address	Dump information for a device node at a specific address.
service-name	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, Visual SoftICE dumps the root device node. If you specify an address, Visual SoftICE dumps information about the device node at that address. If you specify a service name, or a partial name, Visual 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, arbiters, 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.

```
SI>DEVNODE -c e000000086f956f0
DEVICE_NODE
Address           Child             LastChild         InstancePath                ServiceName
------------------------------------------------------------------------------------------------
e000000086dc2010  0000000000000000  0000000000000000  Root\MEDIA\MS_MMVID          audstub
e000000086dc2560  0000000000000000  0000000000000000  Root\MS_PSCHEDMP\0002        PSched
...
e000000086f94b30  0000000000000000  0000000000000000  Root\dmio\0000               dmio
e000000086f950c0  e000000086f8e740  e000000086f8e740  Root\ACPI_HAL\0000
```

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

```
SI>DEVNODE -a e000000086f956f0
ARBITER
Type          Name     Event             DeviceObject      RefCount
---------------------------------------------------------------------
2; Interrupt  RootIRQ  e000000086dc87a0  0000000000000000  0
RANGE
Starting Address  Ending Address    Attributes  Flags      Owner             Owner
------------------------------------------------------------------------------------
2b1               2b1               0           1; SHARED  e000000086f8e350  ACPI
...
```

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

```
SI>DEVNODE -p 80e3d9e0
PARENT TREE
Address   Child     LastChild  InstancePath
                                                                                       ServiceName

----------------------------------------------------------------------------------------------------------
--------------------------------------
80e7a4c8  80eb3ee8  80e764c8   HTREE\ROOT\0

80eb3ee8  80eafc00  80eafc00   Root\ACPI_HAL\0000

...
80e3d9e0  00000000  00000000   IDE\CdRomPHILIPS_CDD4801_CD-R/
RW_________________C2_1____\33373837303648344c57414556432020_0_0_0_0  cdrom
```

See Also

ARBITER, DEVICE, DRIVER

DISCONNECT

Disconnect from the target machine.

Syntax

```
DISCONNECT
```

Use

Use the DISCONNECT command to disconnect from the target machine.

Example

The following example disconnects from the target machine:

```
SI>DISCONNECT
```

See Also

CLOSE, CONNECT, NETFIND, OPEN, WCONNECT

DPC

Display delayed procedure calls.

Syntax

```
DPC [address]
```

address Location of a delayed procedure call.

Use

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

If you provide the address of a specific delayed procedure call, Visual SoftICE displays detailed information for that delayed procedure call.

Example

The following command displays a listing of all DPCs currently in the system:

```
SI>DPC
Address           Number  Importance  DeferredRoutine   DeferredContext   Lock
-------------------------------------------------------------------------------------------
e00000008321c160  0       1           e0000000830073e0  0000000000000000  e000000081a498a8
e000000086c0c8c8  0       1           e0000165e453ae30  e000000086c0c800  e000000081a498a8
```

The following command displays detailed information on the DPC at the address e00000008321c160:

```
SI>DPC e00000008321c160
----------------------------------------------------------------------------
Address         :  ntoskrnl!KiTimerExpireDpc (e00000008321c160)
Number          :  0
...
SystemArgument2 :  0000000000000000
Lock            :  e000000081a498a8
```

See Also

APC

DRIVER

Display information on Windows NT/XP drivers.

Syntax

```
DRIVER [-d] [driver-name | pdriver-object]
```

-d	Dump list of device objects created by the driver.
driver-name	Object directory name of the driver.
pdriver-object	Object address of the driver.

Use

The DRIVER command displays information on Windows NT/XP drivers. If the DRIVER command is entered without parameters, summary information is shown for all drivers found in the operating system Driver Object Collection 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 *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`.

If you use the *-d* flag, Visual SoftICE dumps a list of the device objects created by the driver.

Output

The following fields are shown as detailed information:

DrvName	Name of the driver.
Address	Address of the driver object.
FirstDev	Address of the first device.
DrvFlags	Bit-mask of the driver flags.
DrvLoad	Base address of the driver image.
Size	Size of the driver image.
DrvEntry	Address of the DriverEntry.

StartIo	Address of the driver's StartIo routine.
AddDev	Address of the driver's AddDevice routine.
DrvCreate	Address of the IRP_MJ_CREATE handler.
DrvClose	Address of the IRP_MJ_CLOSE handler.
DrvRead	Address of the IRP_MJ_READ handler.
DrvWrite	Address of the IRP_MJ_WRITE handler.
DrvDevCntrl	Address of the IRP_MJ_DEVICE_CONTROL handler.
DrvIntDevCntrl	Address of the IRP_MJ_INTERNAL_DEVICE_CONTROL handler.
DrvQueryInfo	Address of the IRP_MJ_QUERY_INFORMATION handler.
DrvSetInfo	Address of the IRP_MJ_SET_INFORMATION handler.
DrvQueryEa	Address of the IRP_MJ_QUERY_EA handler.
DrvSetEa	Address of the IRP_MJ_SET_EA handler.
DrvUnld	Address of the driver's unload routine.
DrvPower	Address of the IRP_MJ_POWER handler.
DrvSysCntrl	Address of the IRP_MJ_SYSTEM_CONTROL handler.
DrvPnp	Address of the IRP_MJ_PNP handler.

The following fields are shown only when the *-d* flag is invoked, and the list of device objects created by the driver is dumped:

RefCount	Device object’s reference count.
Address	Address of a DEVICE_OBJECT structure.
DrvObj	Pointer to the driver object that owns the device object.
NextDev	Pointer to the next device object on the linked list of device objects that were created by the same driver.
AttachDev	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.
AttachedTo	Address of the device to which this device is attached.
DevExt	Pointer to device driver-defined device object extension data structure.
DevName	Name of the device, if it has one.

Example

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

```
SI>DRIVER
Address           AddDev            StartIo           DrvFlags  DrvLoad           Size      DrvName
-----------------------------------------------------------------------------------------------------
e00000008618b910  0000000000000000  0000000000000000  12        e0000165e5e3c000  22500     NDProxy
...
e0000000868e2f70  0000000000000000  0000000000000000  52        e0000165e63ea000  10780     VgaSave
e0000000868e4940  0000000000000000  0000000000000000  12        e0000165e679c000  7080      NdisTapi
```

The following is an example of the DRIVER command with the NDIS driver object's name as a parameter. From the listing it can be seen that the driver's first device object is at `e000000086b1d060`.

```
SI>DRIVER ndis
-----------------------------------------------------------------------------------------
DrvName          : NDIS
Address          : e000000086b1d060
...
DrvSysCntrl      : NDIS!.ndisDispatchRequest (e0000165e554b320)
DrvPnp           : NDIS!.ndisDispatchRequest (e0000165e554b320)
```

The following example shows the output of the DRIVER command with the -d flag invoked. This results in dumping a list of the device objects created by the driver.

```
SI>DRIVER -d 80d117f8
DRIVER DEVICE OBJECTS
RefCount   Address    DrvObj     NextDev    AttachDev  AttachedTo  DevExt     DevName
-----------------------------------------------------------------------------------
0          80d0e3f8   80d117f8   80d10030   80d0f020   80d0e8c8    80d0e4b0   CdRom1
0          80d10030   80d117f8 00000000   80d109f8   80e5eab8    80d100e8   CdRom0
```

See Also

ARBITER, DEVICE, DEVNODE

E

Edit memory.

Syntax

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

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

address The address of memory to edit.

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 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
```

A valid floating point number is -1.123456 E-19

Example

The following example moves the null terminated ASCII string "Test String" into memory at location `DS:1000h` on an IA32 target:

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

ERESOURCE

Display information about the synchronization resources contained in ExpSystemResourceList.

Syntax

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

-a	Display resources that are actively held by any thread.
-c	Display resources that are or have been under contention (where the contention count is greater than 0).
-w	Display resources that have threads currently waiting on them.
address	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, Visual SoftICE displays summary information about every ERESOURCE structure in ExpSystemResourceList.

Example

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

```
SI>ERESOURCE address
```

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

```
SI>ERESOURCE -w
```

See Also

KEVENT, KSEM, THREAD

EVAL

Evaluate an expression.

Syntax

```
EVAL expression
```

expression	The expression to evaluate.

Use

To evaluate an expression, enter the EVAL command followed by the expression to evaluate. Visual SoftICE displays the result in decimal, hexadecimal, ASCII, and binary.

You can explicitly evaluate the decimal or hexadecimal values regardless of the current radix setting by using `dec()` or `hex()` with the number in the parenthesis. The Expression Evaluator returns the corresponding value.

Example

The following example displays the hexadecimal, decimal, ASCII, and binary representations of the value of the expression 10*4+3:

```
SI>EVAL 10*4+3
0000000000000043 (67) "C" 0100 0011
```

The following command explicitly evaluates three different expressions using dec() and hex():

```
SI>? dec(111)
0000006f (111)
SI>? hex(111)
00000111 (273)
SI>? dec(111) + hex(111)
00000180 (384)
```

See Also

?, SET EE_EVAL_ORDER, SET EE_IMPL_DEREF, SET RADIX

EXEC

Start a process on the target.

Note: The target must be running for this command to succeed.

Syntax

```
EXEC [program-name]
```

program-name	Path and filename of the process to start on the target.

Use

Use the EXEC command to start a process on the target machine so you can debug it.

Example

The following example starts the process `MyProcess.exe` on the target:

```
SI>EXEC MyProcess.exe
```

See Also

$, KILL, SET GLOBALBREAK, SVCSTART, SVCSTOP

EXIT

Close the current command page or force an exit of the Visual SoftICE master application.

Syntax

```
EXIT [*]
```

Use

The EXIT command exits the current CMD page in the GUI, or the entire master application if the Asterisk (*) parameter is passed.

Example

The following command causes the current CMD page to close:

```
SI>EXIT
```

The following command causes the Visual SoftICE master application to exit:

```
SI>EXIT *
```

See Also

QUIT

EXP

Display export symbols from DLLs.

Syntax

```
EXP [image!]export-name | [!]
```

image! Optional image name to specify the symbol table to display exports from.

export-name A valid export-name. Wildcard characters are fully supported.

! Display list of modules for which Visual SoftICE has exports loaded.

Use

Use the EXP command to show exports from DLLs, and drivers, for which Visual SoftICE has exports loaded.

The image and name parameters can be used to selectively display exports only from the specified image, and/or exports that match the characters and wildcards in the export-name parameter. If you use the EXP command with the ! option, Visual SoftICE displays all tables for which exports are loaded. If you use EXP with only the asterisk (*) wildcard, Visual SoftICE displays all exports in the current table. When exports are displayed, the image name is printed first on a line by itself, and the export names and their addresses are printed below it.

This command is valid for both 32 and 64-bit DLLs with 32-bit exports being listed first.

Example

The following example displays all tables for which exports are loaded.

```
SI>EXP !
ntoskrnl.exe
```

The following example displays all exports in the current table (ntoskrnl.exe).

```
SI>EXP *
8050f718 Export    RtlMoveMemory
8050fa89 Export    RtlPrefetchMemoryNonTemporal
8050fa9a Export    RtlUshortByteSwap
8050faaa Export    RtlUlongByteSwap
8050faba Export    RtlUlonglongByteSwap
...
```

The following example displays all exports for the table ntoskrnl.exe.

```
SI>EXP ntoskrnl.exe!*
8050f718 Export    RtlMoveMemory
8050fa89 Export    RtlPrefetchMemoryNonTemporal
8050fa9a Export    RtlUshortByteSwap
8050faaa Export    RtlUlongByteSwap
8050faba Export    RtlUlonglongByteSwap
...
```

The following example displays all exports for the table ntoskrnl.exe that begin with KeInsert.

```
SI>EXP ntoskrnl.exe!KeInsert*
804d140c Export    KeInsertHeadQueue
804e9e7b Export    KeInsertQueueDpc
804eac80 Export    KeInsertQueue
804eba72 Export    KeInsertDeviceQueue
804ec505 Export    KeInsertByKeyDeviceQueue
804ecf6e Export    KeInsertQueueApc
```

See Also

GETEXP, SET EXPORTPATH, SYM, TABLE

F

Fill memory with data.

Syntax

```
F [-p] address L length data-list
```

-p	Physical address (default is Virtual).
address	Starting address at which to begin filling memory.
L length	Length in bytes.
data-list	List of bytes or quoted strings separated by commas or spaces. A quoted string can be enclosed with single quotes or double quotes.

Use

Memory is filled with the series of bytes (or characters) specified in the data list. Memory is filled starting at the specified address and continues for the 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 to fill the length.

Example

The following example fills memory starting at location `DS:8000h` on an IA-32 target for a length of 100h bytes with the 'test' string. The string 'test' is repeated until the fill length is exhausted.

```
SI>F DS:8000 l 100 'test'
```

FAULTS

Control fault trapping.

Syntax

```
FAULTS [all | none | fault name] [on | off]
```

all	Enable fault trapping for all supported faults.
none	Disable all fault trapping.
fault name	Name of specific supported fault.
on	Enable fault trapping. Only valid when fault name is specified (not with *all* or *none*).
off	Disable fault trapping. Only valid when fault name is specified (not with *all* or *none*).

Use

Use the FAULTS command to control fault trapping. You can select specific supported faults, or select ALL to enable fault trapping for all faults, or NONE to disable fault trapping for all faults. If you select ALL or NONE the on/off switch is ignored. FAULTS only accepts one specific fault to be set at each command. Entering FAULTS without any parameters displays a list of the faults and their states when there is a target connection. The following faults are supported:

ACCESS_VIOLATION

BREAKPOINT

CONTROL_BREAK

CONTROL_C

CPP_EH_EXCEPTION

DATATYPE_MISALIGNMENT

ILLEGAL_INSTRUCTION

IN_PAGE_IO_ERROR

INTEGER_DIVIDE_BY_ZERO

INTEGER_OVERFLOW

INVALID_HANDLE

INVALID_LOCK_SEQUENCE

INVALID_SYSTEM_SERVICE

PORT_DISCONNECTED

SINGLE_STEP

STACK_OVERFLOW

USER_BREAKPOINT

WAKE_SYSTEM_DEBUGGER

WX86_BREAKPOINT

WX86_SINGLE_STEP

Example

The following example enables fault trapping for user breakpoint faults:

```
SI>FAULTS user_breakpoint on
```

The following example enables fault trapping for all faults:

```
SI>FAULTS all
```

FGET

Get a file from a target.

Note: The target must be running for this command to succeed.

Syntax

```
FGET remotefile localfile
```

remotefile	Source path and name of the file on the target.
localfile	Destination path and name to save the file as (on the master).

Use

Use the FGET command to get a remote file from the target and save it to the master.

Example

The following example gets the file `MyFile.exe` on the target and saves it as `MyTargetFile.exe`:

```
SI>FGET MyFile.exe MyTargetFile.exe
```

See Also

FPUT

FIBER

Dump a fiber data structure.

Syntax

```
FIBER [-r | -s] [address]
```

-r	Dump fiber registers.
-s	Dump fiber stack.
address	The address of the fiber data structure.

Use

Use the FIBER command to dump a fiber data structure as returned by the operating system call `CreateFiber()`. Use the *-r* flag to dump the fiber registers. Use the *-s* flag to dump the fiber stack. If you do not specify an address, FIBER dumps the fiber data associated with the current thread. Visual SoftICE provides a stack trace after the dump.

Example

The following example dumps the x86 fiber data associated with the current thread.

```
SI>FIBER
----------------------
Address     : 0014b6d0
User Data   : 001430d0
SEH Pointer : ffffffff
StackTop    : 00630000
StackBtm    : 0062f000
Stack Limit : 00530000
```

The following example dumps the x86 fiber data associated with the registers.

```
SI>FIBER -r
----------------------
Address     : 0014b6d0
User Data   : 001430d0
SEH Pointer : ffffffff
StackTop    : 00630000
StackBtm    : 0062f000
Stack Limit : 00530000
Count: 4
Name  Value
----------------
eip   77e997c8
esp   62fffc
ip    97c8
sp    fffc
```

The following example dumps the IA-64 fiber data associated with the current thread.

```
SI>FIBER
------------------------------
Address     : 000006fbff69d360
User Data   : 000006fbff694838
SEH Pointer : ffffffffffffffff
StackTop    : 000006fbff260000
StackBtm    : 000006fbff25c000
Stack Limit : 000006fbfee60000
```

The following example dumps the IA-64 fiber data associated with the registers.

```
SI>FIBER -r
------------------------------
Address     : 000006fbff69d360
User Data   : 000006fbff694838
SEH Pointer : ffffffffffffffff
StackTop    : 000006fbff260000
StackBtm    : 000006fbff25c000
Stack Limit : 000006fbfee60000
Count: 17
Name         Value
-------------------------------
ip           77cbfce0
gp           9804c0270033f
r5           77ca2020
...
x86.ebp      ff25fff0
x86.csd      3000
```

FILE

Change or display the current source file.

Syntax

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

*	Wildcard character.
file-name	Source file name.

Use

The FILE command is often useful when setting a breakpoint on a line that has no associated symbol. Use FILE to bring the desired file into an SRC page, 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 SRC page.
- If you do not specify *file-name*, the name of the current source file, if any, displays.
- If you specify the * (asterisk), all files in the current symbol table display.

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

Example

The following command displays the file in the SRC page starting with line 1.

```
SI>FILE main.c
```

See Also

ADDSYM, BPX, DELSYM, FS, GETEXP, LOAD, RELOAD, SS, UNLOAD

FMUTEX

Display information about a FASTMUTEX kernel object.

Syntax

```
FMUTEX expression
```

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

Use

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

You must enter an expression to get data, since this is not itself a Windows NT/XP 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 that contains operators, such as `explorer + 0`), or a register name.

Example

The following example displays information about the address contained in register 32:

```
SI>FMUTEX r32
-----------------------------------------------------------
Address    : ntoskrnl!PspJobListLock (e0000000831fb480)
Count      : 0
Owner      : e0000165e61b3fc0
Contention : 0
Event      : ntoskrnl!PspJobListLock+18 (e0000000831fb498)
OldIrql    : 0
```

See Also

KMUTEX, Using a Fast Mutex in the on-line help

FOBJ

Display information about a file object.

Syntax

```
FOBJ fobj-address
```

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

Use

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

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

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

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

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

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

Example

The following example shows output from the FOBJ command:

```
SI>FOBJ e000000086690200
------------------------------------------------------------
Address           :  e000000086690200
DeviceObject      :  e000000086d949e0
...
Event             :  e000000086690298
CompletionContext :  0000000000000000
```

FPUT

Put a file onto a target.

Note: The target must be running for this command to succeed.

Syntax

```
FPUT localfile remotefile
```

localfile Source path and file name of the file on the master.

remotefile Destination path and file name to save the file as on the target.

Use

Use the FPUT command to put a local file from the master onto the target.

Note: The FPUT command has special behavior when used in an AUTOCOPY script. During the AUTOCOPY phase, the copy is being done by a driver doing kernel mode APIs, and not a Ring 3 user application. The format for hard drive locations during the AUTOCOPY phase is:

```
\??\Drive-Letter:\Path\Filename.ext
```

Where the `\??\` is not optional. Without the `\??\` the FPUT command will fail.

Example

The following example puts the file `readme.txt` on the target as part of an AUTOCOPY script:

```
FPUT c:\temp\readme.exe "\??\c:\program files\mysoftware\readme.txt"
```

Note: The quotes are used around the destination path because of white space in the "program files" name, not due to the `\??\` characters.

The following example copies a driver from the master to the target as part of an AUTOCOPY script:

```
FPUT "e:\fs\ext2fs\objchk_wxp_x86\i386\ext2fs.sys" \??\d:\windows\system32\drivers\ext2fs.sys
```

See Also

FGET, SET AUTOCOPYSCRIPT

FS

Search a directory path, and its subdirectories, for a specific file.

Note: The target must be running for this command to succeed.

Syntax

```
FS [-s] Drive:\DirectoryPath\Filename
```

-s Search all subdirectories of the path also.

Drive The letter designating the drive containing the directory path.

DirectoryPath The path to the directory where you want to search for the specified file.

Filename The name of the file for which you are searching. The wildcards *, +, and ? are allowed.

Use

Use the FS command to search a directory for a specified file. Using FS without the -*s* parameter searches only within the directory named, and excludes its subdirectories. Using the -*s* parameter searches in all the subdirectories of the designated directory path as well.

Example

The following example searches for `myfile.txt` on drive D in the `mystuff` directory only:

```
SI>FS D:\mystuff\myfile.txt
```

The following example searches for `myfile.txt` on drive D in the `mystuff` directory and all subdirectories:

```
SI>FS -s D:\mystuff\myfile.txt
```

The following example searches for all files of type .exe on drive D in the `mystuff` directory and all subdirectories:

```
SI>FS -s D:\mystuff\*.exe
```

See Also

FILE, SS

GDT

Display the Global Descriptor Table.

Syntax

```
GDT [-nr] [-all] [selector]
```

-nr	Removes RPL from the selector number display.
-all	Displays all table entries, including illegal or reserved entries.
selector	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 you use GDT with the *-all* option, Visual SoftICE displays all table entries it knows of, including illegal and reserved entries.

Visual SoftICE normally includes the Requestor Privilege-Level (RPL) in its calculation of the selector number. If you wish to remove RPL from the selector number calculation, use the *-nr* option, and Visual SoftICE displays the selector number without the RPL.

On AMD64 and x86 platforms Visual SoftICE displays the table based on the stopped CPU. The target must be stopped before executing the GDT command. On AMD64 platforms there are additional 64-bit descriptor types that Visual SoftICE decodes.

Note: For IA64, Visual SoftICE bases the GDT on a specific 32-bit process. The GDT command will fail unless the current context is on a 32-bit process. Use the ADDR command to switch processes before executing GDT if the current context is not 32-bit.

Output

The base linear address, limit, and count of the GDT are shown at the top of the output. Each subsequent line of the output contains the following information:

Selector	The selector number.
Type	The fully-decoded selector type.
Address	The linear base address of the selector.

Limit The selector's segment size (Granularity indicates scale).

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

Granularity The scaling of the segment limit information (Byte or Page).

Present The selector's present bit, P or NP, indicating whether the selector is present or not present.

Example

The following example illustrates the use of the GDT command on an x86 platform.

```
SI>GDT
Global Descriptor Table - Base Address: 8003f000, Limit: 3ff
Count: 24
Selector  Type                                           Address   Limit     DPL  Granularity  Present
------------------------------------------------------------------------------------------------------
8         Code: Execute/Readable (accessed)              0         ffffffff  0    Page         P
10        Data: Read-Write (accessed)                    0         ffffffff  0    Page         P
1b        Code: Execute/Readable (accessed)              0         ffffffff  3    Page         P
23        Data: Read-Write (accessed)                    0         ffffffff  3    Page         P
28        32bit TSS (busy)                               80042000  20ab      0    Byte         P
...
100       Data: Read-Write (accessed)                    f908a040  ffff      0    Byte         P
108       Data: Read-Write (accessed)                    f908a040  ffff      0    Byte         P
110       Data: Read-Write (accessed)                    f908a040  ffff      0    Byte         P
```

The following example illustrates the use of the GDT command with the *-all* option on an x86 platform.

```
SI>GDT -all
Global Descriptor Table - Base Address: 8003f000, Limit: 3ff
Count: 127
Selector  Type                                           Address   Limit     DPL  Granularity  Present
--------------------------------------------------------------------------------------------------------
0         reserved (Illegal)                             0         0         0    Byte         NP
8         Code: Execute/Readable (accessed)              0         ffffffff  0    Page         P
10        Data: Read-Write (accessed)                    0         ffffffff  0    Page         P
1b        Code: Execute/Readable (accessed)              0         ffffffff  3    Page         P
23        Data: Read-Write (accessed)                    0         ffffffff  3    Page         P
28        32bit TSS (busy)                               80042000  20ab      0    Byte         P
...
3e0       reserved (Illegal)                             0         0         0    Byte         NP
3e8       reserved (Illegal)                             0         0         0    Byte         NP
3f0       reserved (Illegal)                             0         0         0    Byte         NP
```

The following example illustrates the use of the GDT command on an AMD64 platform.

```
SI>GDT
Global Descriptor Table - Base Address: fffff80000343000, Limit: 5f
Count: 7
Selector  Type                               Address           Limit     DPL  Granularity  Present
----------------------------------------------------------------------------------------------------
10        Code: Execute/Readable (accessed)  0                 0         0    NA           P
18        Data: Read-Write (accessed)        0                 0         0    Byte         P
23        Code: Execute/Readable             0                 ffffffff  3    Page         P
2b        Data: Read-Write (accessed)        0                 ffffffff  3    Page         P
33        Code: Execute/Readable (accessed)  0                 0         3    NA           P
40        64bit TSS (busy)                   fffff80000344060  68        0    NA           P
53        Data: Read-Write (accessed)        fffb0000          fff       3    Byte         P
```

The following example illustrates the use of the GDT command with the *-all* option on an AMD64 platform.

```
SI>GDT -all
Global Descriptor Table - Base Address: fffff80000343000, Limit: 5f
Count: 10
Selector  Type                               Address           Limit     DPL  Granularity  Present
----------------------------------------------------------------------------------------------------
0         reserved (Illegal)                 0                 0         0    NA           NP
8         reserved (Illegal)                 0                 0         0    NA           NP
10        Code: Execute/Readable (accessed)  0                 0         0    NA           P
18        Data: Read-Write (accessed)        0                 0         0    Byte         P
23        Code: Execute/Readable             0                 ffffffff  3    Page         P
2b        Data: Read-Write (accessed)        0                 ffffffff  3    Page         P
33        Code: Execute/Readable (accessed)  0                 0         3    NA           P
38        reserved (Illegal)                 40600068          0         0    NA           NP
40        64bit TSS (busy)                   fffff80000344060  68        0    NA           P
53        Data: Read-Write (accessed)        fffb0000          fff       3    Byte         P
```

GETEXP

Get exports from the target machine and add them to the local export cache.

Syntax

```
GETEXP [-s] image-name
```

-s	Search subdirectories.
image-name	The name of the image file.

Use

Use GETEXP to add exports to the local export cache. The root of the local export cache is specified by the EXPORTPATH variable. Visual SoftICE automatically loads exports from the local export cache when a symbol table is loaded. These exports stay loaded until you specifically remove them, or Visual SoftICE exits.

Use the GETEXP command in conjunction with the SET EXPORTPATH command, which sets a destination directory on the master for a local cache of export information extracted from the target. After setting the export path, issue the GETEXP command to retrieve exports from the target and place them in the local export cache. Once exports are stored in the local export cache, Visual SoftICE will automatically load them anytime symbols are not found.

If no path is specified, the system directory (`.exe`) or `system\drivers` directory (`.sys`) is used to find the file on the target. You can retrieve and cache exports for every executable in the system directory of a target by using the *-s* flag with the asterisk (*) wildcard.

Note: Retrieving all exports for a system may take several minutes to complete and cannot be interrupted.

Example

The following example adds exports from an image file to the local export cache:

```
SI>GETEXP myfile.exe
Retrieving exports...
Retrieved exports:
Image: ntoskrnl.exe TimeStamp: 3B7DE38F (45456 bytes)
Cached in local symbol store under 'c:\kayak\exports'.
```

See Also

ADDSYM, DELSYM, FILE, LOAD, RELOAD, SET EXPORTPATH, UNLOAD

GO

Go to an address.

Syntax

```
GO [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

If you specify *break-address*, a single one-time execution breakpoint is set on that address. In addition, all sticky breakpoints are enabled.

Execution begins at the current Instruction Pointer (IP) unless you supply the *start-address* parameter. If you supply the *start-address* parameter, execution begins at that *start-address*. If you attempt to set a *start-address* that is outside the current function scope and the warning level is not set to *off*, then Visual SoftICE generates a warning message asking you to confirm the new *start-address*.

Execution continues until the *break-address* is encountered, or a sticky breakpoint is triggered. When the target stops, for any reason, the one-time execution breakpoint is cleared.

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

Example

The following command sets a one-time breakpoint at address `CS:80123456h` on an IA-32 target:

```
SI>GO 80123456
```

HBOOT

Do a hard system boot (total reset).

Syntax

```
HBOOT
```

Use

The HBOOT command performs a hard reset of the system. It is the same as pressing the Reset button on the computer. It does not shutdown the operating system gracefully.

HBOOT is sufficient unless an adapter card requires a power-on reset. In those rare cases, the machine power must be cycled.

Example

The following command forces the system to reboot.

```
SI>HBOOT
```

See Also

REBOOT, SHUTDOWN, STOP

HEAP

Display the Windows heap.

Syntax

```
HEAP [[-w -x -s -b] [heap | heap-entry | process-type]]
```

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

Use

All HEAP 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.

The 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.

The 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.

The 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.

The 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.

Output

Process	Process that owns the heap.
Heap Base	Base address of the heap, that is, the heap handle.
Id	Heap ID.
Committed	Amount of committed memory used for heap entries.
Present	Amount of present memory used for heap entries.
Reserved	Amount of reserved memory used for heap entries.

SegmentCount Number of heap segments within the heap.

Flags Heap flags, for example, HEAP_GROWABLE (0x02).

Mapped Indicates whether or not the heap is mapped into the process.

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

Base This is the address of the heap entry.

Type Type of the heap entry.

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

Size Size of the heap-entry. Typically, this is the number of bytes available to the application for data storage.

Seg# Heap segment in which the heap-entry is allocated.

Flags Heap entry flags.

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

Segment# Segment number of the heap segment.

Address Range Linear address range that this segment maps to.

Committed Amount of committed memory for this heap segment.

Present Amount of present memory for this heap segment.

Reserved Amount of reserved memory for this heap segment.

Max UCR Maximum uncommitted range of linear memory. This value specifies the largest block that can be created within this heap segment.

Example

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

```
SI>HEAP
Count: 130
Process        Heap Base  Id    Committed  Present  Reserved  SegmentCount  Flags     Mapped
--------------------------------------------------------------------------------------------
smss.exe       00160000   1     6          5        100       1             00000002  no
smss.exe       00260000   2     6          2        10        1             00001002  no
csrss.exe      00160000   1     2b         24       100       1             00000002  no
csrss.exe      00260000   2     6          2        10        1             00001002  no
...
```

The following example displays base address and segment information on a specific heap address.

```
SI>HEAP -b -s 00080000
Process       Heap Base  Id    Committed  Present   Reserved  SegmentCount  Flags     Mapped
--------------------------------------------------------------------------------------------
explorer.exe  00080000   1     9a         85        100       1             00000002  no

Heap SegmentsTable Summary : 00080640

Segment#  Address Range      Committed  Present   Reserved  Max UCR
----------------------------------------------------------------------
0         00080000-00180000  9a         85        100       62000
```

HELP

Display help information.

Syntax

```
HELP [command]
```

```
H [command]
```

command Visual SoftICE command name.

Use

The HELP command displays help on Visual SoftICE commands. (Refer to the ? command for information about evaluating expressions.) To display a list of the available Visual SoftICE commands, enter the HELP command with no parameters. To see detailed information about a specific command, use the HELP command followed by the name of the command on which you want help. Help displays a description of the command, and the command syntax.

Example

The following example displays information about the BC command:

```
SI>HELP BC
Clear a list of, or all breakpoints.
Usage: BC [n | n1 n2 n3 | *]
```

HWND

Display information on Window handles.

Syntax

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

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

Use

The HWND command enumerates and displays information about window handles.

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

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:

Handle HWND handle (refer to OBJTAB for more information). Each window handle is indented to show its child and sibling relationships to other windows.

Class Registered class name for the window, if available (refer to CLASS for more information).

WinProc Address of the message callback procedure. This value is displayed as an address.

TID Owning thread ID.

Image Owning image name (if available). If the image name is unknown, the image handle will be displayed as an address.

Example

The following example uses the HWND command without parameters or options. It will enumerate all the windows in the system, for all desktops.

```
SI>HWND
Count: 97
Handle                Class Name                     WindowProc  Tid       Module Name
------------------------------------------------------------------------------------------
10002                 #32769                         bf8624b8    20c       win32k.sys
10014                #32769                         bf8624b8    20c       win32k.sys
1009a               CiceroUIWndFrame               5fc2e238    788       MSUTB.dll
1009e               CiceroUIWndFrame               5fc2e238    788       MSUTB.dll
200d6               LOGON                          010013fc    1dc
300be               WindowsScreenSaverClass        01001a3b    1dc       logon.scr
10084               BaseBar                        7758cd4b    718       SHELL32.dll
10086              MenuSite                       7758cd4b    718       SHELL32.dll
10088             SysPager                       7195db22    718       SHELL32.dll
1008a            ToolbarWindow32                7196bc2d    718       SHELL32.dll
...
1001c                 #32769                         bf8624b8    23c       win32k.sys
10050                #32769                         bf8624b8    23c       win32k.sys
10052                Message                        bf86248a    23c       win32k.sys
10054               #32768                         bf833a66    23c       win32k.sys
...
```

Note: The output from this example enumerates two desktop windows (handles 10002 and 1001c), 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.

Because the system can 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 OBJTAB DESKTOP command to enumerate all existing desktops in the system.

The following is an example of using the HWND command with a specific window handle and the *-c* option.

```
SI>HWND -c 1001c
Count: 16
Handle                Class Name             WindowProc  Tid       Module Name
---------------------------------------------------------------------------------
1001c                 #32769                 bf8624b8    23c       win32k.sys
10050                #32769                 bf8624b8    23c       win32k.sys
10052                Message                bf86248a    23c       win32k.sys
10054               #32768                 bf833a66    23c       win32k.sys
10036                #32769                 bf8624b8    23c       win32k.sys
100b8               SSDP Server Window     74c124ac    7cc
10038                Message                bf86248a    23c       win32k.sys
...
```

The following is an example of enumerating only those windows owned by thread 718.

```
SI>HWND 718
Count: 26
Handle                Class Name             WindowProc  Tid       Module Name
---------------------------------------------------------------------------------
10084               BaseBar                7758cd4b    718       SHELL32.dll
10086              MenuSite               7758cd4b    718       SHELL32.dll
10088             SysPager               7195db22    718       SHELL32.dll
1008a            ToolbarWindow32        7196bc2d    718       SHELL32.dll
10078               tooltips_class32       719cbaa8    718       comctl32.dll
...
40034               OleMainThreadWndClass  771c9755    718       ole32.dll
```

The following is an example of enumerating those windows owned by the explorer process.

```
SI>HWND explorer
Count: 48
Handle                Class Name             WindowProc  Tid       Module Name
---------------------------------------------------------------------------------
10084               BaseBar                7758cd4b    718       SHELL32.dll
10086              MenuSite               7758cd4b    718       SHELL32.dll
10088             SysPager               7195db22    718       SHELL32.dll
1008a            ToolbarWindow32        7196bc2d    718       SHELL32.dll
10080               tooltips_class32       719cbaa8    6d4       comctl32.dll
10078               tooltips_class32       719cbaa8    718       comctl32.dll
...
100b4               OleMainThreadWndClass  771c9755    7b8       ole32.dll
```

The following is an example of enumerating those windows owned by the Desktop (#32769) class.

```
SI>HWND #32769
Count: 9
Handle                Class Name  WindowProc  Tid       Module Name
---------------------------------------------------------------------
10002                 #32769      bf8624b8    20c       win32k.sys
10014                #32769      bf8624b8    20c       win32k.sys
10004                #32769      bf8624b8    20c       win32k.sys
1000c                #32769      bf8624b8    20c       win32k.sys
1001c                 #32769      bf8624b8    23c       win32k.sys
10050                #32769      bf8624b8    23c       win32k.sys
10036                #32769      bf8624b8    23c       win32k.sys
1002e                #32769      bf8624b8    23c       win32k.sys
1001e                #32769      bf8624b8    23c       win32k.sys
```

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

The following example shows the output when a window handle is specified.

```
SI>HWND 1007a
---------------------------------------------------------------------------
Handle      : 1007a
CLASS       : Application Private : Progman
Module      : 3b7dfe8e1f000adb; SHELL32.dll
WindowProc  : SHELL32!CDesktopBrowser::DesktopWndProc (7741501b)
Title       : Program Manager
Parent      : bc6306e8
Next        : 00000000
1st Child   : bc63ae28
Style       : 96000000; WS_POPUP,WS_VISIBLE,WS_CLIPSIBLINGS,WS_CLIPCHILDREN
ExStyle     : c0000880; WS_EX_TOOLWINDOW
Window Area : 0, 0, 1024, 768 (1024 x 768)
Client Area : 0, 0, 1024, 768 (1024 x 768)
```

Note: If the extended (*-x*) option is specified with a window handle, the same output is generated.

The following example enumerates the windows owned by thread 718 when the extended option (-*x*) is used.

```
SI>HWND -x 718
Count: 26
----------------------------------------------------------------------------------
Handle      : 10084
CLASS       : Application Private : BaseBar
Module      : 3b7dfe8e1f000adb; SHELL32.dll
WindowProc  : SHELL32!DirectUI::VerticalFlowLayout::BuildCacheInfo+14387c
(7758cd4b)
Title       :
Parent      : bc6306e8
Next        : bc63afe0
1st Child   : bc63b338
Style       : 86400000; WS_POPUP,WS_CLIPSIBLINGS,WS_CLIPCHILDREN,WS_CAPTION
ExStyle     : 188; WS_EX_TOPMOST,WS_EX_TOOLWINDOW,WS_EX_WINDOWEDGE
Window Area : 0, 0, 100, 100 (100 x 100)
Client Area : 0, 0, 100, 100 (100 x 100)
----------------------------------------------------------------------------------
Handle      : 10086
CLASS       : Application Private : MenuSite
Module      : 3b7dfe8e1f000adb; SHELL32.dll
WindowProc  : SHELL32!DirectUI::VerticalFlowLayout::BuildCacheInfo+14387c
(7758cd4b)
Title       :
Parent      : bc63b1e8
Next        : 00000000
1st Child   : bc63b448
Style       : 50000000; WS_CHILD,WS_VISIBLE
ExStyle     : ?
Window Area : 3, 3, 3, 3 (0 x 0)
Client Area : 3, 3, 3, 3 (0 x 0)
----------------------------------------------------------------------------------
...
```

See Also

OBJTAB

I

Input a value from an I/O port.

Syntax

```
I[size] port
```

size	**Value**	**Description**
	B	Byte
	W	Word
	D	DWORD
	Q	QWORD

port Port address.

Use

You use the I command to read and display a value from a specified hardware port. Input can be done in byte, word, DWORD, or QWORD lengths. 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 might not be the same as the virtualized data that an application would see.

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

Example

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

```
SI>I 21
```

I1HERE

Stop on embedded INT 1 instructions.

Syntax

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

Use

Use the I1HERE command to specify that any embedded interrupt 1 instruction stops the target. This feature is useful for stopping your program in a specific location. When I1HERE is on, Visual SoftICE checks to see whether an interrupt is really an INT 1 in the code before stopping. If it is not an INT 1, Visual SoftICE will not stop.

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 stops the target. At this point, the current IP 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. Because these tools rely on INT 3 instructions for breakpoint notifications, I1HERE allows you to use INT 1 instructions as hardwired interrupts in your code without triggering the application debugger.

Example

The following example enables I1HERE mode. Any INT 1 instructions generated after this point stop Visual SoftICE.

```
SI>I1HERE on
```

See Also

I3HERE, SET

I3HERE

Stop 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 INT 3 instruction stops Visual 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 instruction occurs, it stops the target. At this point, the current IP is the instruction after the INT 3 instruction.

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

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 BoundsChecker, you should place INT 1 instructions in your code instead of INT 3 instructions. Refer to I1HERE.

Example

The following example turns on I3HERE mode. Any INT 3 instructions generated after this point cause the target to stop.

```
SI>I3HERE on
```

When the command I3HERE is set to ON, and you are using a Ring 3 debugger, such as BoundsChecker, Visual SoftICE traps on any INT 3 breakpoints installed by the Ring 3 debugger.

See Also

I1HERE, SET

IMAGE (MOD)

Display the operating system image list. The MOD command remains aliased to the IMAGE command.

Syntax

```
IMAGE [-l] [-u | -s] | [partial-name]
MOD [-l] [-u | -s] | [partial-name]
```

-l	Displays the images in load-order.
-u	Displays only images in user space.
-s	Displays only images in system space.
partial-name	Prefix of the image name. Accepts wildcards.

Use

This command displays the operating system image list. If a partial name is specified, only images that begin with the name will be displayed. Visual SoftICE displays images in the following order:

32- and 64-bit driver images (Windows NT only)

32- and 64-bit application images

The IMAGE command is process-specific. All images will be displayed that are visible within the current process. This includes all 32- and 64-bit images and all driver images. This means if you want to see specific images, you must switch to the appropriate address context before using the IMAGE command.

Output

For each loaded image the following information is displayed:

address	Base linear address of the executable file.
size	Hex value representing the size of the image file in bytes.
name	Name specified in the .DEF file using the NAME or LIBRARY keyword.
fullname	Full path and file name of the image's executable file.

Example

The following example shows how the IMAGE command displays images:

```
SI>IMAGE
96 images
        Address  Size              Name           FullName
---------------------------------------------------------------------------------------------
-------
20000001ff000000  546b80            win32k         \??\C:\WINDOWS\system32\win32k.sys
...
e0000165e6c2c000  1c80              RDPCDD         \SystemRoot\System32\DRIVERS\RDPCDD.sys
e0000165e6c76000  1100              dxgthk         \SystemRoot\System32\drivers\dxgthk.sys
```

The following example displays the output from the IMAGE command using the *-u* parameter:

```
SI>IMAGE -u
13 images
        Address  Size              Name       FullName
---------------------------------------------------------------------------------------------
0000000000400000  20000             siservice  C:\Program Files\NuMega\SoftICE Distributed
Edition\siservice.exe
0000000060a20000  1fa000            DBGHELP    C:\WINDOWS\system32\DBGHELP.dll
...
0000000077e80000  17c000            ntdll      C:\WINDOWS\System32\ntdll.dll
```

See Also

ADDR, PROCESS, THREAD

IMAGEMAP (MAP32)

Display a memory map of all modules currently loaded in memory. The MAP32 command remains aliased to the IMAGEMAP command.

Syntax

```
IMAGEMAP [image-name | address]
```

image-name	Windows image-name.
address	Any address that falls within an executable image.

Use

Using IMAGEMAP with no parameters lists information about all images.

If you specify either an *image-name* as a parameter, only sections from the specified image are shown. For each image, one line of data is printed for every section belonging to the image.

Because the IMAGEMAP command takes any address that falls within an executable image, an easy way to see the memory map of the image that contains the current IP is to enter:

```
IMAGEMAP .
```

IMAGEMAP 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 IMAGEMAP's output contains the following information:

Owner	Image name.
Name	Section name from the executable file.
Obj#	Section number from the executable file.
Address	Selector:offset address of the section.
Size	Section’s size in bytes.
Type	Type and attributes of the section, as follows:

Type	**Attributes**
CODE	Code
IDATA	Initialized Data
UDATA	Uninitialized Data
RO	Read Only
RW	Read/Write
SHARED	Object is shared

Example

The following example illustrates sample output for IMAGEMAP.

```
SI>IMAGEMAP
cmd    .text 0001 000000004ad02000 0005fcd8 20
cmd   .pdata 0002 000000004ad62000 0000126c 40
cmd  .srdata 0003 000000004ad64000 000009e4 40
...
ntdll    .rsrc 0006 0000000077fcc000 0002c0e8 40
ntdll   .reloc 0007 0000000077ffa000 0000403a 40
```

INTOBJ

Display information on system interrupt objects.

Syntax

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

vector	The interrupt vector associating the hardware interrupt and the object.
interrupt-object-address	The address for a specific interrupt object.

Use

The INTOBJ command displays information about interrupt objects that are current in the system. If you enter INTOBJ without parameters, Visual 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, Visual SoftICE displays information about the specified interrupt object.

Example

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

```
SI>INTOBJ
Address           Vector    Irql  SynchIrql  Shared  SrvRoutine        IntMode
---------------------------------------------------------------------------------
e0000000867a8f40  51        5     b          no      e0000165e4d0ea20  1
e000000086d12ad0  61        6     6          yes     e0000165e1dd5380  0
...
e000000086c21e80  a2        a     a          yes     e0000165e1fc40e0  0
e0000000867a8db0  b2        b     b          no      e0000165e4cfa9a0  1
```

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

```
SI>INTOBJ 61
----------------------------------------------------------------------
Address    :  e000000086d12ad0
...
IntMode    :  0; LevelSensitive
```

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

```
SI>INTOBJ e000000086f269f0
----------------------------------------------------------------------
Address    :  e000000086f269f0
...
IntMode    :  1; Latched
```

IRP

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

Syntax

```
IRP -f | -n | -p | -a | irp-address
```

-f	Display all IRP stack locations.
-n	Display the next IRP stack location
-p	Walk the previous IRP stack location
-a	Iterates through all threads on a system and shows the IRP for each thread
irp-address	Address of the start of the IRP structure to be displayed

Use

The IRP command displays the contents of the I/O Request Packet and the contents of associated current I/O stack located at the specified address. Note that the command does not check the validity of the IRP structure at the specified address, so any address will be accepted by Visual SoftICE as an IRP address. Be careful to pass the IRP command a valid IRP address.

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

Flags	**Flags used to define IRP attributes.**
StackCount	The number of stack locations that have been allocated for the IRP. A common device driver bug is to access non-existent stack locations, so this value can be useful in determining when this has occurred.
CurrentLocation	This number indicates which stack location is the current one for the IRP. Again, this value, combined with the previous StackCount, can be used to track down IRP stack-related bugs.
Tail.Overlay.CurrentStackLoc	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.

Cancel

This boolean is set to TRUE if the IRP has been cancelled as a result of an IRP cancellation call. An IRP can 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 might be useful:

Major Function, Minor Function

These fields indicate what type of request the IRP is being used for. The major function is used in determining which request handler will be called when an IRP is received by a device driver. The minor function provides the specifics about the request.

Device Object

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.

File Object

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

Completion Routine

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

Example

The following example shows the output for the IRP command:

```
SI>IRP e000000086f898f0
----------------------------------------------------------------------
Address                           :  e000000086f898f0
Type                              :  6
Size                              :  118
...
Tail.CompletionKey                :  0000000000000000
----------------------------------------------------------------------
Address          :  e000000086f899c0
MajorFunction    :  e
...
Context          :  0000000000000000
```

IT (IDT)

Display the Interrupt Table. The IDT command remains aliased to the IT command.

Syntax

```
IT [-c cpu] [interrupt-number]
IDT [-c cpu] [interrupt-number]
```

-c cpu	Specify the CPU.
interrupt-number	The number of the interrupt to display details about.

Use

The command displays the interrupt table of the operating system, or details about a given interrupt if you specify an interrupt-number. On x86, the interrupt table is called the Interrupt Descriptor Table (IDT). On IA64, the interrupt table is called the Interrupt Vector Area (IVA).

Output

The following fields are shown as detailed information:

Vector Interrupt vector number.

SrvRoutine Address and/or symbolic name for the interrupt service routine.

The following are output only for x86 platforms:

Type Type of interrupt:
Task Gate
32-bit Trap Gate
16-bit Trap Gate
32-bit Interrupt Gate
16-bit Interrupt Gate
32-bit Call Gate
16-bit Call Gate

Present Indicates whether the entry is present or not.

DPL Interrupt descriptor privilege level (DPL), which is either 0, 1, 2, or 3.

The following is output only for AMD64:

IST Displays the value of the AMD64 Interrupt Stack Table (IST).

Examples

In the following example, the IT command displays the interrupt descriptor table for a specific vector on x86:

```
SI>IT 13
Interrupt Descriptor Table - CPU 0, Base Address: 8003f400, Limit: 7ff
----------------------------------------------
Vector     : 13
Type       : e; 32bit Interrupt Gate
Present    : yes
DPL        : 0
SrvRoutine : 0008:804d80c8 (ntoskrnl!KiTrap13)
```

In the following example, the IT command displays the interrupt descriptor table of the operating system on x86:

```
SI>IT
Interrupt Descriptor Table - CPU 0, Base Address: 8003f400, Limit: 7ff
Count: 256
Vector    Type                     Present  DPL  SrvRoutine
------------------------------------------------------------------------------------------------
0         e; 32bit Interrupt Gate  yes      0    0008:f91d2dcc (BCHKD!_Section_LDATA+204c)
1         e; 32bit Interrupt Gate  yes      3    0008:804d5b06 (ntoskrnl!KiTrap01)
2         5; Task Gate             yes      0    0058:8053d306 (ntoskrnl!g_rgAttributeTags+f6)
3         e; 32bit Interrupt Gate  yes      3    0008:804d5e2e (ntoskrnl!KiTrap03)
...
fd        e; 32bit Interrupt Gate  yes      0    0008:804d4c82 (ntoskrnl!KiUnexpectedInterrupt205)
fe        e; 32bit Interrupt Gate  yes      0    0008:804d4c89 (ntoskrnl!KiUnexpectedInterrupt206)
ff        e; 32bit Interrupt Gate  yes      0    0008:804d4c90 (ntoskrnl!KiUnexpectedInterrupt207)
```

In the following example, the IT command displays the interrupt descriptor table of the operating system on IA-64:

```
SI>IT
Interrupt Descriptor Table - CPU 0, Base Address: e0000000831a0000
Count: 32
Offset    Type                              SrvRoutine
------------------------------------------------------------------------------------------------
0         0; VHPT                           e0000000831a0000 (ntoskrnl!KiIvtBaseILog)
400       0; Instruction TLB                e0000000831a0400 (ntoskrnl!.KiInstTlbVectorILog)
800       0; Data TLB                       e0000000831a0800 (ntoskrnl!.KiDataTlbVectorILog)
c00       0; Alternate Instruction TLB      e0000000831a0c00 (ntoskrnl!.KiAltInstTlbVectorILog)
1000      0; Alternate Data TLB             e0000000831a1000 (ntoskrnl!.KiAltDataTlbVectorILog)
...
```

KDLIST

List loaded kernel debugger extension DLLs in search-order.

Syntax

```
KDLIST
```

Use

Use KDLIST to list the kernel debugger extension DLLs loaded on the target. The list is presented in the order they are discovered while searching.

Example

The following example shows the output from KDLIST:

```
SI>KDLIST
Name    Version     Type   Path                                                           CreateTime

----------------------------------------------------------------------------------------------------------
-----------
kext    4.0.18.0    1.0.0  e:\program files\Debugging Tools for Windows\winext\kext.dll   Mon Dec 17
13:22:57 2001
kdexts  5.1.3591.0  1.0.0  e:\program files\Debugging Tools for Windows\winxp\kdexts.dll  Mon Dec 10
17:10:02 2001
```

KEVENT

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, Visual 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, Visual SoftICE displays information about the specified event.

Example

The following example shows how to use the KEVENT command to display kernel events that are currently in the system:

```
SI>KEVENT
Address           Type  SignalState  Name

----------------------------------------------------------------------------------------------------------
----------------------------------
e00000008635dcb0  1     0            StopUtilityManagerEvent

...
e0000000864ec630  1     0            WINMGMT_COREDLL_LOADED
```

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

```
SI>KEVENT e000010600689d18
----------------------------------------------------------------------
Address      :  e000010600689d18
Type         :  0; Notification
SignalState  :  0
WaitListHead :  0012019f00000000
Name         :  RotHintTable
```

See Also

KMUTEX, KSEM

KILL

Terminate a process on the target.

Note: The target must be running for this command to succeed.

Syntax

```
KILL [pid]
```

pid The process ID.

Use

Use the KILL command to terminate a process on the target.

Example

The following example terminates process 3c8 on the target:

```
SI>KILL 3c8
```

See Also

EXEC, SET GLOBALBREAK, SVCSTART, SVCSTOP

KMUTEX

Display information about kernel mutexes.

Syntax

```
KMUTEX [kernel-mutex]
```

kernel-mutex	Kernel mutex address

Use

If you issue the KMUTEX command without any parameters, Visual 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, Visual SoftICE displays information about the kernel mutex at that address.

Example

The following example shows how the KMUTEX command is used to display information about a specific object:

```
SI>KMUTEX e0000000864646c0
---------------------------------------------------------------
Address      :  e0000000864646c0
...
Name         :  RasPbFile
```

See Also

FMUTEX, KEVENT, KSEM

KOBJECT

Display information about Kernel Objects a thread can wait on, or a list of threads waiting on an object.

Syntax

```
KOBJECT [-w] address
```

-w Display a list of threads waiting on the specified object.

address The address of the object.

Use

Use KOBJECT to display information about kernel objects a thread can wait on. Use the -*w* flag to display a list of threads waiting on the specified object.

Example

The following example shows information about the kernel objects a thread is waiting on:

```
SI>KOBJECT ffb76da8
-------------------------
Address      : ffb76da8
Type         : 3; Process
SignalState  : 0
WaitListHead : 80d48e18
```

The following example shows the list of threads waiting on the specified object:

```
SI>KOBJECT -w ffb76da8
Address   Thread    Object    Type     WaitKey   WaitType
-------------------------------------------------------------
80d48e18  80d48da8  ffb76da8  Process  0         1
```

KSEM

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, Visual 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, Visual 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:

```
SI>KSEM e0000000864d86b0
----------------------------------------------------------------------
Address      :  e0000000864d86b0
Limit        :  7fffffff
SignalState  :  0
WaitListHead :  e0000000864d86b8
Name         :  shell.{090851A5-EB96-11D2-8BE4-00C04FA31A66}
```

See Also

KEVENT, KMUTEX

LOAD

Load symbols.

Syntax

```
LOAD [image-name]
```

image-name Name of the image or symbol file (`.exe` or `.pdb`) for which you want to load symbols (you can use the * wildcard).

Use

Use the LOAD command to load symbols for an image file. If the symbols do not apply to any current process image they will not be loaded, because symbols are dynamic in Visual SoftICE. If you use the * wildcard in place of an image-name, Visual SoftICE opens the Symbol Files utility. Use the ADDSYM command for persistently loaded symbols.

Example

The following example loads symbols for the `myprogram.exe` image file:

```
SI>LOAD myprogram.exe
```

See Also

ADDSYM, DELSYM, FILE, GETEXP, RELOAD, SET SYMSRVSEARCH, SET SYMTABLEAUTOLOAD, UNLOAD

LOCALS

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. Note that expandable items are delineated with a plus (+) sign.

Example

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

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

See Also

TYPES, WL

LOG

Echo the input, output, or both to a file.

Syntax

```
LOG off | input | output | all [-o] [filename]
```

off	Disable logging to a file.
input	Log input to Visual SoftICE to a file.
output	Log output from Visual SoftICE to a file.
all	Log both input and output to a file.
-o	Overwrite the specified file.
filename	Specify the file to which Visual SoftICE will echo the log.

Use

Use the LOG command to echo the input and/or output of a console or Command page to a file. The console where you execute the command will echo to the specified file in the manner you specified. The first time you use LOG you must specify a *filename*. When you use LOG with *off* to turn off the logging, you do not specify the filename. Turning off logging does not un-set the declared file for that console or Command page, so when you turn on logging for that console or Command page again the new log will be appended to the existing log file unless you declare a new filename. Passing the *-o* switch instructs Visual SoftICE to overwrite the contents of the specified filename with the output.

If you have multiple consoles or Command pages open you can echo them all to different files, or to the same file. However, you cannot control in what order the operating system will commit I/O to the file.

To view the current LOG setting, use the LOG command without any parameters.

Example

The following example enables output logging for a specific console or Command page and specifies the file as `file.txt`, and also instructs

Visual SoftICE to overwrite the current contents of file.txt with the new output:

```
SI>LOG output -o file.txt
Logging Output (file.txt) [Overwrite]
```

The following example disables logging for a specific console or Command page, but leaves the output file declared:

```
SI>LOG off
Logging off (file.txt)
```

The following example re-enables all logging for a specific console or Command page when the output file was previously declared:

```
SI>LOG all
Logging All (file.txt) [Overwrite]
```

See Also

@, SAVE, SCRIPT

M

Move data.

Syntax

```
M [-p] source-address L length dest-address
```

-p	Specify physical memory address. The default is a virtual memory address.
source-address	Start of address range to move.
L length	Length in bytes in hexadecimal.
dest-address	Start of destination address range.

Use

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

Example

The following command moves 8192 bytes (expressed in hexadecimal) from memory location 80d9c5b0h to 80db2a28h on an IA32 target.

```
SI>M 80d9c5b0  l 2000 80db2a28
```

MACRO

Define a new command that is a collection of Visual SoftICE commands.

Syntax

MACRO [-d] [*macro-name*] | [*] | [= “*macro-body*”]

-d	Delete the specified macro.
macro-name	Case-insensitive name for the macro being defined, or the name of an existing macro.
*	Wildcard symbol to specify all defined macros when using *-d* to delete.
=	Define (or redefine) a macro.
macro-body	Quoted string that contains a list of Visual SoftICE commands and parameters separated by semi-colons (;).

Use

The MACRO command is used to define new keywords that are collections of existing Visual SoftICE commands. Defined macros can be executed directly from the Visual SoftICE command line. The MACRO command is also used to list or delete individual macros.

If no options are provided, a list of all defined macros will be displayed.

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 contain any alphanumeric character. If the *macro-name* parameter specifies an existing macro, the MACRO command redefines the existing macro. The *macro-name* parameter cannot be a duplicate of an existing "real" Visual SoftICE command. The *macro-name* parameter must be followed by an equal sign “=”, which must be followed by the quoted string that defines the *macro-body* parameter.

The *macro-body* parameter must be embedded between beginning and ending quotation marks. The *macro-body* parameter is made up of a collection of existing Visual 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* parameter 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* parameter, precede the character with a backslash character (\). Because the backslash character is used for escape sequences, to specify a literal backslash character, use two consecutive backslashes (\\). The final command within the *macro-body* parameter does not need to be terminated by a semi-colon.

Note: A *macro-body* parameter cannot be empty. It must contain one or more non-white space characters. A *macro-body* parameter 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 limited to 32 iterations. 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:

```
SI>MACRO
XWHAT =
"WHAT EAX;WHAT EBX;WHAT ECX; WHAT EDX; WHAT ESI; WHAT EDI"

OOPS =
"I3HERE OFF;GENINT 3"

1shot =
"bpx eip do \"bc bpindex \""
```

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

The following examples show other basic uses for the MACRO command:

`SI>MACRO -d *`	Delete all defined macros.
`SI>MACRO -d oops`	Delete the macro named oops.
`SI>MACRO`	Display all defined macros.

The following example is a simple macro definition:

```
SI>MACRO help = “h”
```

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

```
SI>MACRO help = “h exp”
```

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

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

```
SI>MACRO help= “help %1”
```

In the revised example, an optional parameter was defined to pass to the Visual 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 valid macro definitions:

```
SI>MACRO qexp = “addr explorer; query %1”
SI>MACRO 1shot = “bpx %1 do \”bc bpindex\””
SI>MACRO ddt = “dd thread”
SI>MACRO ddp = “dd process”
```

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 Visual SoftICE command name. Visual SoftICE commands cannot be redefined.

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.

MAP32 (IMAGEMAP)

Display a memory map of all modules currently loaded in memory. The MAP32 command remains aliased to the IMAGEMAP command.

Syntax

```
IMAGEMAP [image-name | address]
```

image-name	Windows image-name.
address	Any address that falls within an executable image.

Use

Using IMAGEMAP with no parameters lists information about all images.

If you specify either an *image-name* as a parameter, only sections from the specified image are shown. For each image, one line of data is printed for every section belonging to the image.

Because the IMAGEMAP command takes any address that falls within an executable image, an easy way to see the memory map of the image that contains the current IP is to enter:

```
IMAGEMAP .
```

IMAGEMAP 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 IMAGEMAP's output contains the following information:

Owner	Image name.
Name	Section name from the executable file.
Obj#	Section number from the executable file.
Address	Selector:offset address of the section.
Size	Section’s size in bytes.
Type	Type and attributes of the section, as follows:

Type	**Attributes**
CODE	Code
IDATA	Initialized Data
UDATA	Uninitialized Data
RO	Read Only
RW	Read/Write
SHARED	Object is shared

Example

The following example illustrates sample output for MAP32.

```
SI>MAP32
cmd    .text 0001 000000004ad02000 0005fcd8 20
cmd   .pdata 0002 000000004ad62000 0000126c 40
cmd  .srdata 0003 000000004ad64000 000009e4 40
...
ntdll    .rsrc 0006 0000000077fcc000 0002c0e8 40
ntdll   .reloc 0007 0000000077ffa000 0000403a 40
```

MSR

Read, write, or enumerate model-specific registers.

Syntax

MSR *index* [=*value*]
MSR *name* [=*value*]

index	Register index.
name	Register name.
=value	Value to write to the register.

Use

Use the MSR command to read, write, or enumerate model-specific registers. Some processors do not support any MSRs. If you do not specify a register name or index, Visual SoftICE enumerates known MSRs for the target. If you supply only the register *index* or *name*, Visual SoftICE reads the value currently in that MSR. If you specify an *index* or *name* along with a *value*, Visual SoftICE writes the value to the register. The target must be stopped when reading and writing MSRs.

Example

The following example enumerates the MSRs on a Pentium II processor:

```
SI> MSR
Count: 60
Index     Name                 Description                      Value

-------------------------------------------------------------------------------------
0         IA32_P5_MC_ADDR      MachineCheck Exception Address   0
1         IA32_P5_MC_TYPE      MachineCheck Exception Type      0
10        IA32_TIME_STAMP_CTR  Time Stamp Counter               3cc350b803db
17        IA32_PLATFORM_ID     Platform ID                      2042000000000000
...
40a       IA32_MC2_ADDR        MC2_ADDR                         3446ff003446ff
40c       IA32_MC3_CTL         MC3_CTL                          1
40d       IA32_MC3_STATUS      MC3_STATUS                       0
```

Note: The Value column is only displayed when the target is stopped.

The following example reads from the Platform ID MSR on a Pentium II, whose index is 17:

```
SI> MSR 17
(17) IA32_PLATFORM_ID = 2042000000000000
```

The following example writes FFFF to DEBUGCTLMSR on a Pentium III, whose index is 1d9:

```
SI> MSR 1d9 = FFFF
```

See Also

R, RG

NAME

Show, add, or delete user-defined variables (such as constants and addresses).

Syntax

NAME [-a | -d] [*name* | *] [*value*]

-a	Assign a user-defined address.
-d	Delete the specified user-defined variable.
name	The name of the variable, constant, or address.
*	Wildcard used with *-d* to delete all user-defined variables.
value	The value you are assigning to the name.

Use

Use the NAME command to display, add, or delete user-defined variables (constants and addresses). Using NAME without any parameters displays a list of all the user defined variables. Using NAME with *-a* assigns a user-defined address. Using NAME with *-d* allows you to delete a specific user-defined variable, or all user-defined variables using the * wildcard.

Example

The following example displays a list of the user-defined variables:

```
SI>NAME
    _ProcessListHead = address : e00000008321f180 size: 8
    ...
   _IdleProcessBlock = address : e000000081a6fa40 size: 8
```

The following example assigns a user-defined constant TEMP, and then shows that it has had the correct value assigned to it:

```
SI>NAME TEMP 234
SI>name
    _ProcessListHead = address : e00000008321f180 size: 8
   ...
   _IdleProcessBlock = address : e000000081a6fa40 size: 8
                TEMP = value : 0000000000000234 (564)
```

The following example assigns a user-defined address to ADDR, and then shows that it has had the correct value assigned to it:

```
SI>NAME ADDR 0x3c6
SI>name ADDR
                ADDR = value : 00000000000003c6 (966)
```

The following example deletes the TEMP user-defined constant, and then displays the list to show that it has been removed:

```
SI>NAME -d TEMP
SI>name
    _ProcessListHead = address : e00000008321f180 size: 8
  ...
                ADDR = value : 00000000000003c6 (966)
```

The following example deletes all user-defined variables:

```
SI>NAME -d *
```

NETFIND

Display a list of accessible target machines.

Syntax

```
NETFIND [-dns]
```

-dns Enable DNS lookup.

Use

Use the NETFIND command to display a list of accessible target machines. You can then try to connect to the target you need using the information displayed. The NETFIND command also returns the operating system build number and operating system version string for each target found.

Example

The following example displays a list of valid target machines on the system with DNS lookup enabled:

```
SI>NETFIND -dns
                         Agent                                           Sicore   Current
Ip Address       Name           Active  Cpu          Os Version                     Version  State

----------------------------------------------------------------------------------------------------------
----------------------------
255.255.255.0  JSPILLANE        yes     IA32(x86)-2  Windows NT/XP v5.1 Build:2600  1.0.470  (run )
Listening....
255.255.254.0  JASPILLANE       yes     IA32(x86)-1  Windows NT/XP v5.0 Build:2195  1.0.471  (run )
Listening....
255.255.253.0  POWERBOX         yes     IA32(x86)-1  Windows NT/XP v5.0 Build:2195  1.0.471  (run )
Listening....
255.255.252.0  GANDALF          yes     IA64-?       Windows XP-64 v?.? Build:????  ?.?.?    (run )
Listening....
255.255.251.0  HOLLYWOOD        yes     IA64-1       Windows XP-64 v5.1 Build:3604  1.0.459  (run )
Listening....
...
```

See Also

CLOSE, CONNECT, DISCONNECT, OPEN, WCONNECT

NTCALL

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 a wrapper for routines in NTOSKRNL, where the real work is done at level 0. If you use Visual SoftICE to step through one of these calls, you will see that Windows will perform a processor-specific instruction to transition between a privilege level 3 API and a privilege level 0 routine that actually implements the call. The index number of the function is passed in the EAX register on x86 platforms, and the R8 register is used on IA64 platforms.

If you want 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 the TABLE command.

Output

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

Index	Hexadecimal index number of the function passed in EAX.
Address	Selector:offset address of the start of the function.
Parameters	Number of DWORD parameters passed to the function.
Symbol	Either the symbolic name of the function, or the offset within NTOSKRNL if no symbols are loaded.

Example

The following example shows the output from NTCALL on an x86 target:

```
SI>NTCALL
Service table address:80544c00  Number of services:276
Index     Address   Parameters  Name
--------------------------------------------------------------------------------------------
0         8058391a  6           ntoskrnl!NtAcceptConnectPort
1         8056b154  8           ntoskrnl!NtAccessCheck
2         80560664  b           ntoskrnl!NtAccessCheckAndAuditAlarm
...
119       8061cc87  4           ntoskrnl!NtReleaseKeyedEvent
11a       8061cf0e  4           ntoskrnl!NtWaitForKeyedEvent
11b       80605e85  0           ntoskrnl!NtQueryPortInformationProcess
```

The following example shows the output from NTCALL on an IA-64 target:

```
SI>NTCALL
Service table address:e0000000831f6c20  Number of services:284
Index     Address           Parameters  Name
-------------------------------------------------------------------------------------------------------
0         e000000083313e00  ???         ntoskrnl!.NtAcceptConnectPort
1         e0000000833fea60  8           ntoskrnl!.NtAccessCheck
2         e000000083409ac0  b           ntoskrnl!.NtAccessCheckAndAuditAlarm
...
11b       e0000000833966c0  ???         ntoskrnl!.NtQueryPortInformationProcess
```

Note: The question marks (???) indicate that all the parameters are passed via registers, and that Visual SoftICE was unable to determine the number from the operating system data.

NTSTATUS

Display header-defined mnemonics for NTSTATUS error codes.

Syntax

NTSTATUS *code*

code The NTSTATUS error code you want a mnemonic returned for.

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`:

```
SI>NTSTATUS 0x5e
OBJECT_INITIALIZATION_FAILED
```

O

Output a value to an I/O port.

Syntax

```
O[size] port value
```

size	**Value**	**Description**
	B	Byte
	W	Word
	D	DWORD
	Q	QWORD

port	Port address.
value	Byte, word, DWORD, or QWORD value as specified by the size parameter.

Use

Output to PORT commands are used to write a value to a hardware port. Output can be done in byte, word, DWORD, or QWORD lengths. 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 (Ports 21h and A1h).

Example

The following command performs an output to port 21, which unmasks all interrupts for interrupt controller one.

```
SI>O 21 0
```

OBJDIR

Display objects in a Windows NT 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 with no parameters displays the named objects within the root object directory. To list the objects in a subdirectory, enter the full object directory path.

Output

The following information will be displayed by the OBJDIR command:

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

Example

The following example shows how to use the OBJDIR command to display objects in the root object directory:

```
SI>OBJDIR
Address           Header            Name                      Type
-----------------------------------------------------------------------
e000000086465590  e000000086465560  NLAPrivatePort            WaitablePort
e000000086466c80  e000000086466c50  NLAPublicPort             WaitablePort
...
e000010600a46330  e000010600a46300  SmSsWinStationApiPort     Port
e000010600a77860  e000010600a77830  XactSrvLpcPort            Port
```

See Also

OBJTAB

OBJTAB

Display entries in the WIN32 user object-handle table.

Syntax

```
OBJTAB [-h] [handle | object-type-name | index]
```

-h	Display list of valid object-type-names.
handle	Object handle.
object-type-name	One of the object-type-names, predefined by Visual SoftICE:

FREE	Free handle
HWND	Window handle
HMENU	Menu handle
HCURSOR	Cursor handle
HICON	Icon handle
HDWP	Deferred window position handle
HHOOK	Window hook callback handle
CLIPDATA	Clipboard data handle
QUEUE	Call procedure handle
HACCEL	Accelerator table handle
DDEACCESS	DDE access handle
HCONV	DDE conversion handle
HDDEDATA	DDE data handle
HMONITOR	Display monitor handle
HKL	Keyboard layout handle
HKF	Keyboard layout file handle
HWINEVENTHOOK	Window event hook callback handle
HWINSTA	Window station handle
HIMC	Input context handle
HHID	Human interface device data handle
HDEVINFO	Device information set handle

DESKTOP	Window handle that is a Desktop type window

index	Index value for object-handle.

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, Visual SoftICE lists the full contents of the master object-handle table. If an object handle is specified, just that object is listed. If an *object-type-name* is entered, all objects in the master object-handle table of that type are listed.

Output

The following information is displayed by the OBJTAB command:

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

Examples

The following is an abbreviated example using the OBJTAB command with the *-h* flag set.

```
SI>OBJTAB -h
Count: 21
Id  Type           Description
-------------------------------------------------------
0   FREE           Free handle
1   HWND           Window handle
2   HMENU          Menu handle
...
12  HHID           Human Interface Device Data handle
13  HDEVINFO       Device Information set handle
```

The following is an abbreviated example using the OBJTAB command with a specified type.

```
SI>OBJTAB hicon
Count: 51
Object    Type     Id  Handle    Owner     Description
----------------------------------------------------------
e14b8498  HCURSOR  3   10003     ffb8f020  Cursor handle
e14b8518  HCURSOR  3   10005     ffb8f020  Cursor handle
e14bf9b0  HCURSOR  3   10007     ffb8f020  Cursor handle
...
```

The following is an abbreviated example using the OBJTAB command without any parameters.

```
SI>OBJTAB
Count: 341
Object    Type       Id  Handle    Owner     Description
------------------------------------------------------------------------
00000000  FREE       0   10000     00000000  Free handle
bc5d1ba8  HMONITOR   c   10001     00000000  Display monitor handle
e1c4c758  HWND       1   10002     ffba9020  Window handle
e14b8498  HCURSOR    3   10003     ffb8f020  Cursor handle
...
```

The following example uses OBJTAB to enumerate all existing desktops in the system.

```
SI>OBJTAB desktop
Count: 2
Object    Type  Id  Handle    Owner     Description
-------------------------------------------------------
e1c4c758  HWND  1   10002     ffba9020  Window handle
e1d7f4a0  HWND  1   1001c     ffbc8020  Window handle
```

See Also

OBJDIR

OPEN

Opens a user or kernel crash dump file.

Syntax

```
OPEN filename
```

filename	Path and name of the core-dump or crash file you want to explore.

Use

Use the OPEN command to explore a core-dump or crash file. Assuming the file is reasonable and accurate, opening a connection to it treats it as a target where you can perform any standard *read* actions. This allows you to walk-through and explore the file. Depending on the type of crash dump, you will have access to different data and commands. A full kernel dump gives the most access while a mini user-dump gives the least.

Example

The following example opens a core-dump file as a target:

```
SI>OPEN d:\memory.dmp
Connected to:
        Name           : d:\memory.dmp
       Processor      : IA64-Itanium
       Stepping       : 0
       Processor Count: 2
       Operating Sys. : Windows XP-64 Ver. 5.1 Build 2600
       Target Agent   : Not available.
```

See Also

CLOSE, CONNECT, DISCONNECT, NETFIND, WCONNECT

P

Execute one program step.

Syntax

```
P [RET]
```

RET	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 pointer 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 Visual SoftICE.

If RET is specified, Visual SoftICE will step until it finds a return or return from interrupt instruction.

If the Register page is visible when Visual SoftICE stops, all registers that have been altered since the P command was issued are highlighted. 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 Visual SoftICE is single-stepping every instruction.

The P command, by default, is thread-specific. If the current Instruction Pointer (IP) is executing in thread X, Visual SoftICE will not break until the program step occurs in thread X. This prevents the case of Windows NT process switching or thread switching during the program step causing execution to stop in a different thread or process than the one you were debugging.

While stepping, you can abort the active step by issuing the STOP command, pressing the Stop toolbar button, or pressing **Ctrl-Break**. If the step was issued from the command page, then the red **Abort Command** button will abort it as well.

Note: Whether or not the target responds to the stop that the master issues is dependent upon the state of the target.

Example

The following example executes one program step:

```
SI>P
```

See Also

STOP, T

PACKET

Display the contents of a network packet.

Syntax

```
PACKET [address] [length]
```

address	Address of the network packet.
length	Length of the network packet.

Use

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

Note: Currently, only Ethernet packets are supported.

Output

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

Examples

The following example shows the output of the PACKET command.

```
SI>PACKET r33
ETHERNET
------------------------------------
DestAddr   : ff:ff:ff:ff:ff:ff
SourceAddr : 00:d0:b7:ab:fb:9c
Type       : 0806; ARP request/reply
ARP
-------------------------------------
HwAddrType        : 1
ProtocolAddrType  : 800
HwAddrSize        : 6
ProtocolAddrSize  : 4
Operation         : 1
SenderHwAddr      : 00:d0:b7:ab:fb:9c
SenderIpAddr      : 172.23.101.42
TargetHwAddr      : 00:00:00:00:00:00
Target Ip Address : 172.23.96.3
```

The following example shows the output of the PACKET command after using the SET PACKETFORMAT command to change the format to STRUCTURE.

```
SI>SET PACKETFORMAT STRUCTURE
SI>PACKET r33
NDIS_PACKET
--------------------------------
Address       : e00000008689e540
PhysicalCount : 0
TotalLength   : 0
Head          : e000000086bfdb30
Tail          : e000000086bfdb30
Pool          : 00000000706f6f4c
Count         : 0
NdisFlags     : 2;
ValidCounts   : no
Flags         : 0;
OobDataOffset : 80
Reserved0     : e0000000869d2370
Reserved1     : 0
Reserved2     : 0
Reserved3     : 0
NDIS_BUFFER
------------------------------------------------------------------------
Address        : e000000086bfdb30
Next           : 0000000000000000
Size           : 38
Flags          : c; MDL_SOURCE_IS_NONPAGED_POOL|MDL_ALLOCATED_FIXED_SIZE
Process        : 0000000000000000
MappedSystemVa : e00000008689e630
StartAddress   : e00000008689e000
Count          : 2a
Offset         : 630
```

See Also

SET PACKETFORMAT

PAGE

Display page table information.

Syntax

```
PAGE [address [L num_pages]]
```

address	The virtual address about which you want to know page table information.
L num_pages	The number of pages to display.

Use

You can use the PAGE command to display the top-level page directories or explore the page translation mapping for a particular address. If you use the PAGE command without any parameters, Visual SoftICE displays all the top level page entries. On x86 and AMD64 this is usually one entry. On IA64 this is usually three entries.

Use the PAGE command with a single address to show the entire page translation hierarchy, from top-most table entry to bottom-most table entry, including the physical mapping of the virtual address entered. This is generally two levels deep on x86, and can be up to four levels deep on 64-bit platforms.

Use the PAGE command with an address and the *L num-pages* parameter to display the lowest level page translation hierarchy for each page requested, walking *num_pages* from the specified address. When you specify a number of addresses for decoding, Visual SoftICE adds an additional column (address) for easier interpretation of the results.

About Page Tables

On the x86 platform, a page directory usually 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. 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).

NT 4.0 and Windows 2000 can use the 4 MB page feature of the Pentium/ Pentium Pro processors. NTOSKRNL, HAL, and all boot drivers are mapped into a 4 MB page starting at 2 GB (80000000h).

Output

The PAGE command output contains the following information:

Entry Indicates what translation table (level) the entry is (PXE, PDPE, PDE, PDE-LARGE, or PTE).

Physical Physical address mapped to the entered virtual address. This is only valid on the lowest translation level entry.

Data Actual contents of the translation table entry.

Physical Page (PPN) Start of the translation table the entry is in (the system's physical page number).

Attributes Attributes of the page entry:

P/NP — Present or Not Present

D — Dirty

A — Accessed

S/U — Supervisor/User

R — Read Only

RW — Read-Write

R-EXE — Read and Execute (IA64 only)

RW-EXE — Read-Write and Execute (IA64 only)

EXE-PROMOTE — Execute/Promote (Ring 3 IA64)

G — Global (x86 and AMD64 only)

ED — Exception Deferred (IA64 only)

CACHE-(???) — Cacheable Memory (Cache-type in parenthesis)

WTC — Write-Through Cache

WBC — Write-Back Cache

WC — Write Coalescing Cache (IA64 only)

NaTPage — IA64 Only Caching Mode

Example

The following example shows the PAGE command on an x86 machine.

```
SI>PAGE
Page Size: 0x1000 bytes
Entry  Address   Type
---------------------------
PDE    39000     (Kernel)
```

The following example shows the PAGE command on an IA64 machine.

```
SI>PAGE
Page Size: 0x2000 bytes
Count: 3
Entry  Address   Type
----------------------------
PXE    7002000   (User)
PXE    0         (Session)
PXE    7000000   (Kernel)
```

The following example shows the PAGE command on an AMD64 machine.

```
SI>PAGE
Page Size: 0x1000 bytes
Entry  Address   Type
---------------------------
PXE    374000    (Global)
```

The following example shows the PAGE command specifying an address on an x86 machine.

```
SI>PAGE f9b1d480
Entry         Physical  Data      Physical Page(PPN)  Attributes
------------------------------------------------------------------
PDE           0         1016963   1016000 (1016)      P A RW S
-PTE          31c4480   31c4121   31c4000 (31c4)      P A R S G
```

The following example shows the PAGE command specifying an address on an IA64 machine.

```
SI>PAGE e0000165e3cb71c0
Entry         Physical  Data              Physical Page(PPN)  Attributes
-------------------------------------------------------------------------------------------
PDPE          0         1000000700e661    700e000 (700e)      P A D RW-EXE S CACHE-(WBC) ED
-PDE          0         10000007fc8661    7fc8000 (7fc8)      P A D RW-EXE S ED
--PTE         392d1c0   1000000392c221    392c000 (392c)      P A R-EXE S ED
```

The following example shows the PAGE command specifying an address on an AMD64 machine.

```
SI>PAGE fffffadff1ce9ed0
Entry         Physical  Data              Physical Page(PPN)  Attributes
---------------------------------------------------------------------------
PXE           0         2c00063           2c00000 (2c00)      P A RW S
-PDPE         0         3a73063           3a73000 (3a73)      P A RW S
--PDE         0         3c02163           3c02000 (3c02)      P A RW S
---PTE        60feed0   61300000060fe121  60fe000 (60fe)      P A R S G
```

The following example shows the PAGE command specifying an address and number of pages on an x86 machine.

```
SI>PAGE f9b1d480 L 10
Address       Entry  Physical  Data      Physical Page(PPN)  Attributes
-----------------------------------------------------------------------------
f9b1d480      PTE    31c4480   31c4121   31c4000 (31c4)      P A R S G
f9b1e480      PTE    31c5480   31c5121   31c5000 (31c5)      P A R S G
f9b1f480      PTE    31c6480   31c6963   31c6000 (31c6)      P A D RW S G
...
f9b2a480      PTE    3191480   3191963   3191000 (3191)      P A D RW S G
f9b2b480      PTE    3192480   3192121   3192000 (3192)      P A R S G
f9b2c480      PTE    319a480   319a963   319a000 (319a)      P A D RW S G
```

The following example shows the PAGE command specifying an address and number of pages on an IA64 machine.

```
SI>PAGE e0000165e3cb71c0 L 10
Address           Entry  Physical  Data              Physical Page(PPN)  Attributes
-----------------------------------------------------------------------------------------
e0000165e3cb71c0  PTE    392d1c0   1000000392c221    392c000 (392c)      P A R-EXE S ED
e0000165e3cb91c0  PTE    38af1c0   100000038ae221    38ae000 (38ae)      P A R-EXE S ED
e0000165e3cbb1c0  PTE    39b11c0   100000039b0221    39b0000 (39b0)      P A R-EXE S ED
...
e0000165e3cd11c0  PTE    39471c0   10000003946661    3946000 (3946)      P A D RW-EXE S ED
e0000165e3cd31c0  PTE    38c91c0   69300000038c8221  38c8000 (38c8)      P A R-EXE S ED
e0000165e3cd51c0  PTE    38cb1c0   3f900000038ca221  38ca000 (38ca)      P A R-EXE S ED
```

The following example shows the PAGE command specifying an address and number of pages on an AMD64 machine.

```
SI>PAGE fffffadff1ce9ed0 L 10
Address           Entry  Physical  Data              Physical Page(PPN)  Attributes
--------------------------------------------------------------------------------------
fffffadff1ce9ed0  PTE    60feed0   61300000060fe121  60fe000 (60fe)      P A R S G
fffffadff1ceaed0  PTE    60ffed0   64500000060ff121  60ff000 (60ff)      P A R S G
fffffadff1cebed0  PTE    6280ed0   6280063           6280000 (6280)      P A D RW S
...
fffffadff1cf6ed0  PTE    6174ed0   6174121           6174000 (6174)      P A R S G
fffffadff1cf7ed0  PTE    6175ed0   6175121           6175000 (6175)      P A R S G
fffffadff1cf8ed0  PTE    6176ed0   6176121           6176000 (6176)      P A R S G
```

PCI

Dump/Read/Write the configuration registers for a PCI device in the system.

Syntax

Enumerate PCI Devices

```
PCI
```

Dump Details on Specific Device

```
PCI bus.device.function
```

bus	Bus number
device	Device number
function	Function number

Read LENGTH Bytes of a Device Configuration Space at OFFSET

```
PCI bus.device.function [-b | -w | -d] [offset[L length]]
```

bus	Bus number
device	Device number
function	Function number
-b	Byte format
-w	Word format
-d	D-word format
offset	Function offset
L length	Length of dump

Edit/Write LENGTH Bytes to a Device Configuration Space at OFFSET

```
PCI -e bus.device.function [-b | -w | -d] [offset] data
```

-e	Edit PCI data
bus	Bus number
device	Device number
function	Function number
-b	Byte format
-w	Word format
-d	DWORD format
offset	Function offset
data	Data to write to PCI device

Use

The PCI command acts on the registers for the PCI devices on the system. Using the PCI command you can list all PCI devices on a system, dump the registers, read the registers, or edit the data by writing a value to the registers. 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.

Examples

The following example illustrates the use of the PCI command to display a list of the PCI devices:

```
SI>PCI
Bus       Device    Function  VendorID  DeviceID  Name
----------------------------------------------------------------------------------------------------
0         0         0         8086      7124      Host/PCI Bridge Device
0         1         0         8086      7125      VGA PC Compatible Display Controller
...
1         c         0         10b7      9200      Ethernet Network Controller
```

The following example illustrates the use of the PCI command to read details about a specific PCI device:

```
SI>PCI 0.1e.0
----------------------------------------------------------------------------------
Bus             : 0
Device          : 1e
Function        : 0
VendorID        : 8086
DeviceID        : 2418
...
BIST            : 0
```

The following example illustrates the use of the PCI command to dump 6f bytes of PCI config space in raw byte format starting at the offset of f:

```
SI>PCI 0.1e.0 f L 6f -b
0000000f:  00 00 01 00 00 00 00 00 00 00 00 00 00 01 01 40     ...............@
...
0000006f:  00 00 00 00 40 00 00 00 00 00 00 00 4b 4b 4b     ....@.......KKK
```

The following example illustrates the use of the PCI command to write the data 01010101 in word format to the PCI device, starting at the offset f:

```
SI>PCI -e 0.1e.0 -w f 01010101
```

PEEK

Read from physical memory.

Syntax

```
PEEK[size] address
```

size	**Value**	**Description**
	b	Byte
	w	Word
	d	DWORD
	s	Short Real
	l	Long Real
	t	10-Byte Real
	q	QWORD

Note: 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`:

```
SI>PEEKD FF000000
```

See Also

PHYS, POKE

PHYS

Display all virtual addresses that correspond to a physical address.

Syntax

```
PHYS address
```

address Memory address that the CPU 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 CPU 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 can appear in more than one page table entry, and therefore more than one virtual address range.

The PHYS command is specific to the current address context. It searches the Page Tables and Page Directory associated with the current Visual SoftICE address context.

Example

Physical address `a0000h` is the start of VGA video memory. Video memory often shows up in multiple virtual addresses in Windows. The following example shows three different virtual addresses that correspond to physical `a0000h`.

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

PING

Check the current connection to the target.

Syntax

```
PING
```

Use

Use the PING command to check the current connection to the target machine.

Example

The following example uses the PING command to check the current connection to the target:

```
SI>PING
Target connection ok.
Connected to:
Name            : KLOS-IA64
Processor       : IA64-Itanium
Stepping        : 0
Processor Count: 1
Operating Sys. : Windows NT (64bit) Ver. 5.1 Build 2505
Target Agent    : Connected (Active)
```

POKE

Write to physical memory.

Syntax

```
POKE[size] address value
```

size	**Value**	**Description**
	b	Byte
	w	Word
	d	DWORD
	s	Short Real
	l	Long Real
	t	10-Byte Real
	q	QWORD

Note: Size defaults to b.

address	Physical memory address.
value	Value to write to memory.

Use

POKE writes a byte, word, DWORD, or QWORD 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`:

```
SI>POKED FF000000 12345678
```

See Also

PEEK, PHYS

PROCESS

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

Syntax

```
PROCESS [[-x] [-o] [-m] process-type | thread-ID]
```

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

Use

If you use the PROCESS 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*) for Windows NT. 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, Visual SoftICE uses raw values from within the operating system 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. Because 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:

Process	Process name.

KPEB	Address of the Kernel Process Environment Block.

PID	Process ID.

Threads	Number of threads owned by the process.

Priority	Base priority of the process.

UserTime	Relative amount of time the process spent executing code at the user level.

KrnlTime	Relative amount of time the process spent executing code at the kernel level.

State	Current status of the process:

- Running: The process is currently running.
- Ready: The process is in a ready to run state.
- Idle: The process is inactive.
- Swapped: The process is inactive, and its address space has been deleted.
- Transition: The process is currently between states.
- Terminating: The process is terminating.

Example

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

```
SI>PROCESS -x csrss
------------------------------------------------------------
KPEB                    :  e000000086712e60
...
HandleCount             :  10b
```

The following example uses the objects option (-*o*) to display objects for a process, `fib64_2`:

```
SI>PROCESS -o fib64_2
Handle    Inheritable  ObjHeader         Object            Type       Name
-----------------------------------------------------------------------------------------
0         yes          e000000086570990  e0000000865709c0  Event
4         yes          e00000008666ddf0  e00000008666de20  Event
...
20        yes          e000000086d6ff98  e000000086d6ffc8  Mutant     NlsCacheMutant
24        no           0000000000000000  0000000000000030  None       Empty slot
```

See Also

ADDR, IMAGE, THREAD

QUERY (ADDRESSMAP)

Display the virtual address map of a process.

Syntax

```
QUERY [[-a] address] | [process-type]
```

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

Use

The QUERY command displays a map of 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:

Context — Address context.

Address Range — Start and end address of the linear range.

Flags — Flags from the node structure.

MMCI — Pointer to the memory management structure.

PTE — Structure that contains the ProtoPTEs for the address range.

Name — Additional information about the range. This includes the following:

- Memory mapped files will show the name of the mapped file.
- Executable modules will show the file name of the DLL or EXE.
- Stacks will be displayed as (thread ID).
- Thread information blocks will be displayed as TIB (thread ID).
- Any address that the WHAT command can identify might also appear.

Example

The following example uses the QUERY command to map a specific linear address for Windows NT.

```
SI>QUERY -a 77f50000
Count: 14
Context         Address Range      Flags     MMCI      PTE       Name
-------------------------------------------------------------------------------
System          77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
smss.exe        77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
csrss.exe       77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
winlogon.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
services.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
lsass.exe       77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe     77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe     77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe     77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
svchost.exe     77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
spoolsv.exe     77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
explorer.exe    77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
siservice.exe   77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
logon.scr       77f50000-77ff8000  07100005  80ad8008  e13585e0  ntdll.dll
```

The following example uses the QUERY command to list the address map of the explorer process for Windows NT.

```
SI>QUERY explorer
Address Range      Flags     MMCI      PTE       Name
--------------------------------------------------------------------
00010000-00010000  c4000001
00020000-00020000  c4000001
00030000-0006f000  8400000f
00070000-00070000  01400000  809e5540  e10e5150
00080000-0017f000  840000b1                      Process Heap
...
7ffdc000-7ffdc000  c6400001                      Tib:338
7ffde000-7ffde000  c6400001                      Tib:13c
7ffdf000-7ffdf000  c6400001                      UPEB (20c)
```

QUIT

Close the current command page or force an exit of the Visual SoftICE master application.

Syntax

```
QUIT [*]
```

Use

The QUIT command closes the current command page, or forces an exit of the Visual SoftICE master application.

Example

The following command causes the current command page to close:

```
SI>QUIT
```

The following command causes the Visual SoftICE master application to exit:

```
SI>QUIT *
```

See Also

EXIT

R

Display or change the register values.

Syntax

```
R [-c processor] [-f] [-s] [-d] [register-name [=value]] |
[register-group-name] | [all]
```

-c processor	Specify the CPU number.
-f	Display the register fields.
-s	Display symbol information for the value stored in the register.
-d	Display descriptive text about the register or register field.
register-name	Display the named register.
value	Set the register to the value indicated.
register-group-name	Display the registers for the specified register group.
all	Display all the registers.

Use

If *register-name* is supplied without a value, Visual SoftICE displays the register and its current value.

If both *register-name* and *value* are supplied, the specified register's contents are set to the value indicated.

If the CPU number is specified using the *-c* parameter, the Visual SoftICE displays the registers on that CPU.

If the *-s* parameter is used, Visual SoftICE displays symbol information about the value stored in the register. For example, a register could contain an address to jump to, and supplying *-s* would translate that value into something like a function name or image!section+offset location.

The *-f* parameter displays the register fields.

The *-d* parameter displays descriptive text about the register or register fields. Since most people don't remember what all the fields of a given register are, the *-d* parameter can be very useful. All register and field values are displayed in hex.

For more information on register names and groups, refer to Understanding Register Names and Groups in the on-line help.

Example

The following example sets the r14 register equal to 200h:

```
SI>R r14=200
       r14 0000000000000200
```

The following example displays the registers in the application register group:

```
SI>R application
       ar.kr0 (ar0) 00000ffffc000000        ar.kr1 (ar1) 0000000000000000
...
      ar.pfs (ar64) 0000000000000204        ar.lc (ar65) 0000000000000000
       ar.ec (ar66) 0000000000000000
```

The following example displays the registers for CPU 0:

```
SI>R -c0
                 ip e0000165dc60e850                slot 0
                cfm 8000000000000000           zero (r0) 0000000000000000
 ...
      ar.bsp (ar17) e0000000819f6288  ar.bspstore (ar18) e0000000819f6230
      ar.rsc (ar16) 0000000000580003      ar.rnat (ar19) e0000165e47e40aa
```

The following example displays the FPSR register fields:

```
SI>R -f cfm
       cfm 0000000000000592    cfm.sof 12 (18)             cfm.sol b (11)
   cfm.sor 0                cfm.rrb.gr 0                cfm.rrb.fp 0
cfm.rrb.pr 0
```

The following example displays the symbol information for the IP register:

```
SI>R -s ip
        ip 00000000004022ae fib32_2!fib_func
```

The following example displays descriptive text for the IP register:

```
SI>R -d ip
        ip 00000000004022ae instruction pointer
```

See Also

MSR, RG, SET FLOATREGFORMAT, SET REGNAME, Understanding Register Names and Groups in the on-line help

REBOOT

Reboot the target machine you are currently connected to.

Note: The target must be running for this command to succeed.

Syntax

```
REBOOT
```

Use

Use the REBOOT command to reboot the currently connected target machine.

Example

The following example reboots the target machine:

```
SI>REBOOT
```

See Also

HBOOT, SHUTDOWN, STOP

RELOAD

Reload symbols for an image file.

Syntax

```
RELOAD [image-name]
```

image-name Name of the image for which you want to reload symbols.

Use

Use the RELOAD command to reload symbols for an image.

Example

The following example reloads symbols for the `myprogram.exe` image:

```
SI>RELOAD myprogram.exe
```

See Also

ADDSYM, DELSYM, FILE, GETEXP, LOAD, SET SYMSRVSEARCH, SET SYMTABLEAUTOLOAD, UNLOAD

RG

Display the register group names on the target.

Syntax

```
RG
```

Use

Enter RG to list the register group names available on the target. For more information on register names and groups, refer to Understanding Register Names and Groups in the on-line help.

Example

The following example shows how the RG command displays the register group names on the target:

```
SI>RG
17 register groups

state
general
local
float
rfloat
application
pred
rpred
branch
cpuid
perfdata
system
breakpoint
perfconfig
region
protection
translation
```

See Also

MSR, R, SET REGNAME, Understanding Register Names and Groups in the on-line help

S

Search memory for data.

Syntax

```
S [address L length data-list]
```

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

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.

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 (x86/Windows NT: 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.

```
SI>S ES:DI+10 L ECX 'Hello',12,34
```

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

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

SAVE

Save names and macros.

Syntax

```
SAVE [filename]
```

filename Name of the file to contain your saved names and macros.

Use

Use the SAVE command to save your user-defined names and macros to a file so you can load them during your next debug session. You can load saved names and macros like any other script file.

Example

The following example saves the user-defined names and macros to a file:

```
SI>SAVE mynames.txt
```

See Also

@, SCRIPT, SET LOG

SCRIPT

Load and execute a script file on the target.

Syntax

```
SCRIPT [file-name]
```

file-name	The file name of the script you want to load and execute.

Use

Use the SCRIPT command to load and execute a script file on the target.

Example

The following example loads and executes `MyScript.txt` on the target:

```
SI>SCRIPT MyScript.txt
```

See Also

@, SAVE, SET LOG, SET SCRIPTECHO, SET SCRIPTPATH, SET SCRIPTSTOPONERROR, SLEEP

SET

Display the state of all console and execution flags.

Syntax

```
SET
```

Use

Using SET without parameters displays a list of all the console and execution flags and their current settings.

Example

The following example displays the current console and execution settings:

```
SI>SET
Name                   Status
----------------------------------------------------------------------
ADDRESSFORMAT          SHORT (trims empty high 32bits of 64bit addr)
CODE                   off
...
UPPERCASE              off
WARNLEVEL                LOW
```

See Also

SET ADDRESSFORMAT, SET CACHE, SET DBGMSGDEBOUNCETIME, SET DIALECT, SET EE_EVAL_ORDER, SET EXEPATH, SET EXPORTPATH, SET FLOATREGFORMAT, SET GLOBALBREAK, SET IMAGEMATCH, SET KDEXTPATH, SET LOG, SET MSGLEVEL, SET PACKETFORMAT, SET RADIX, SET REGNAME, SET SCRIPTECHO, SET SCRIPTPATH, SET SCRIPTSTOPONERROR, SET SRCPATH, SET STEPMODE, SET STICKYCONTEXT, SET STOPONCMD, SET SYMPATH, SET SYMSRVSEARCH, SET SYMTABLEAUTOLOAD, SET THREADP, SET UPPERCASE, SET WARNLEVEL

SET ADDRESSFORMAT

Select the format with which to display addresses.

Syntax

```
SET ADDRESSFORMAT [long | short | backquote]
```

long Display all digits of an address (16 digits for a 64-bit address).

short Trim the empty high 32-bits of a 64-bit address when appropriate.

backquote Separate the high and low 32-bits of a 64-bit address with a back quote character, as in KD or Windbg.

Note: None of these options affects the display for 32-bit addresses, they are always 8 digits.

Use

Use the SET ADDRESSFORMAT command to configure the way Visual SoftICE displays addresses.

To view the current ADDRESSFORMAT setting, use the SET ADDRESSFORMAT command without any parameters.

Example

The following example selects the short address format:

```
SI>SET ADDRESSFORMAT short
ADDRESSFORMAT = SHORT (trims empty high 32bits of 64bit addr)
SI>IMAGE -u
Count: 15
      Address  Size      Name      FullName
--------------------------------------------------------------------------
      01000000  2c000     notepad   C:\WINDOWS\system32\notepad.exe
      47350000  98000     UxTheme   C:\WINDOWS\system32\UxTheme.dll
     ...
      77c80000  1e6000    kernel32  C:\WINDOWS\system32\kernel32.dll
      77e70000  18a000    ntdll     C:\WINDOWS\System32\ntdll.dll
```

The following example selects the long address format:

```
SI>SET ADDRESSFORMAT long
ADDRESSFORMAT = LONG (shows all digits of 64bit addr)
SI>IMAGE -u
Count: 15
     Address    Size      Name      FullName
--------------------------------------------------------------------------
0000000001000000  2c000     notepad   C:\WINDOWS\system32\notepad.exe
0000000047350000  98000     UxTheme   C:\WINDOWS\system32\UxTheme.dll
    ...
0000000077c80000  1e6000    kernel32  C:\WINDOWS\system32\kernel32.dll
0000000077e70000  18a000    ntdll     C:\WINDOWS\System32\ntdll.dll
```

The following example selects the back quote separated address format:

```
SI>SET ADDRESSFORMAT backquote
ADDRESSFORMAT = BACKQUOTE (separates high/low 32bits of 64bit addr with `)
SI>IMAGE -u
Count: 15
    Address      Size      Name      FullName
--------------------------------------------------------------------------
00000000`01000000  2c000     notepad   C:\WINDOWS\system32\notepad.exe
00000000`47350000  98000     UxTheme   C:\WINDOWS\system32\UxTheme.dll
   ...
00000000`77c80000  1e6000    kernel32  C:\WINDOWS\system32\kernel32.dll
00000000`77e70000  18a000    ntdll     C:\WINDOWS\System32\ntdll.dll
```

See Also

SET

SET AUTOCOPYSCRIPT

Set the AUTOCOPY script to run when a target requests it at boot time.

Syntax

```
SET AUTOCOPYSCRIPT [filename]
```

filename The name of the AUTOCOPY script file for the target to run.

Use

Use the SET AUTOCOPYSCRIPT command to configure a script to run when the target requests it at boot time. The script should have only FPUT commands in it, and the target must be configured for AUTOCOPY at boot time. To configure the target for AUTOCOPY at boot time, use the DSConfig utility.

Note: The FPUT command has special behavior when used in an AUTOCOPY script. During the AUTOCOPY phase, the copy is being done by a driver doing kernel mode APIs, and not a Ring 3 user application. The format for hard drive locations during the AUTOCOPY phase is:

```
\??\Drive-Letter:\Path\Filename.ext
```

Where the `\??\` is not optional. Without the `\??\` the FPUT command will fail.

You can specify a full path name as part of the script filename, or specify only the filename if the script exists in the path defined by the SET SCRIPTPATH command.

When a target configured for AUTOCOPY boots, it halts the loading of drivers shortly after the filesystem is available and signals the master. The master then runs the designated AUTOCOPY script and signals the target to continue with a normal operating system boot.

This process is intended to allow you to put new files onto the target to be used by the boot process that is underway, and it is done early enough in the boot sequence to support such things as replacement video drivers.

To view the current AUTOCOPYSCRIPT setting, use the SET AUTOCOPYSCRIPT command without any parameters.

Example

The following example sets `targetstartup.txt` as the Auto Copy script. The file `targetstartup.txt` exists in the defined script path.

```
SI>SET AUTOCOPYSCRIPT targetstartup.txt
AUTOCOPYSCRIPT = targetstartup.txt
```

The following example shows a typical AUTOCOPY script.

```
FPUT "e:\fs\ext2fs\objchk_wxp_x86\i386\ext2fs.sys" \??\d:\windows\system32\drivers\ext2fs.sys
```

See Also

FPUT, SET, SET SCRIPTPATH

SET CACHE

Set the size of the cache in KB.

Syntax

```
SET CACHE [size]
```

size	Cache size in KB.

Use

Use the SET CACHE command to set the Visual SoftICE buffer size to cache the virtual memory read.

To view the current CACHE setting, use the SET CACHE command without any parameters.

Example

The following example sets the cache to 1000 KB:

```
SI>SET CACHE 1000
CACHE = 1000
```

See Also

SET

SET DBGMSGDEBOUNCETIME

Control the responsiveness of the UI to Debug Message notifications.

Syntax

```
SET DBGMSGDEBOUNCETIME value
```

value Number of milliseconds.

Use

Use the SET DBGMSGDEBOUNCETIME command to control the UI's responsiveness by managing how long the Master will append multiple debug messages received in a row (debouncing) prior to sending a single notification to the UI with all the appended messages. The default setting is 50 milliseconds, which has been found to be optimal for most cases. Assigning a value of 0 effectively disables this feature and every debug message received will be immediately distributed to the UI.

To view the current DBGMSGDEBOUNCETIME setting, use the SET DBGMSGDEBOUNCETIME command without any parameters.

Example

The following example sets the debounce time to 45 milliseconds:

```
SI>SET DBGMSGDEBOUNCETIME 2d
DBGMSGDEBOUNCETIME = 2d (45) msecs
```

See Also

SET

SET DIALECT

Set the input command dialect.

Syntax

```
SET DIALECT [sic | kd]
```

sic Enable SoftICE Classic command dialect (default).

kd Enable KD command dialect.

Use

Use the SET DIALECT command to indicate whether you want SoftICE Classic or KD input dialect. After executing the command, the display prompt displays the appropriate text (SIC or KD) to indicate which dialect you have configured.

To view the current DIALECT setting, use the SET DIALECT command without any parameters.

Example

The following example enables SoftICE Classic dialect:

```
SI>SET DIALECT sic
DIALECT = sic
```

The following example enables KD dialect:

```
SI>SET DIALECT kd
DIALECT = kd
```

See Also

SET

SET EE_EVAL_ORDER

Select the order of tests performed by the expression evaluator during the parse stage.

Syntax

```
SET EE_EVAL_ORDER [S R D N]
```

S	Symbols
R	Registers
D	Datums
N	Numerics (radix rules still apply)

Use

Use the SET EE_EVAL_ORDER command to set the order of tests performed by the expression evaluator during the parse stage. This command takes a four-letter character string comprised of the letters SRDN as input, where you decide the order of evaluation tests by changing the order of the four letters. The default order is SRDN.

To view the current EE_EVAL_ORDER setting, use the SET EE_EVAL_ORDER command without any parameters.

Example

The following example evaluates "a" with the default testing order of SRDN:

```
SI>? a
```

Visual SoftICE first searches the current symbol table for a symbol with the name "a". If it is not found, it goes on to search the processor's register set for a register, or register alias, named "a". If it is still not found, it goes on to search system datums for a match. Finally, if it fails in the first three tests, Visual SoftICE evaluates "a" as a number. If the radix is set to hex, as is default, then the evaluation comes back as a number with the decimal value of 10.

If the evaluation order is set to NSRD, Visual SoftICE first tries to evaluate "a" as a number. Again, if the radix is set to hex, then the evaluation comes back as a number with the decimal value of 10.

See Also

?, EVAL, SET

SET EE_IMPL_DEREF

Select the expression evaluator's behavior regarding evaluation of expressions containing symbols that are pointers.

Syntax

```
SET EE_IMPL_DEREF [on | off]
```

on	Enable expression evaluator dereferencing.
off	Disable expression evaluator dereferencing.

Use

Use the SET EE_IMPL_DEREF command to control the expression evaluator's behavior regarding dereferencing. If you have set EE_IMPL_DEREF to *on*, and the expression evaluator encounters an expression containing a symbol that is a pointer, it will use the value it points to for evaluation. If you have set EE_IMPL_DEREF to *off*, and the expression evaluator encounters an expression containing a symbol that is a pointer, it will use the address of the pointer for evaluation.

To view the current EE_IMPL_DEREF setting, use the SET EE_IMPL_DEREF command without any parameters.

Example

The following example enables expression evaluator dereferencing, evaluates an expression containing a symbol that is a pointer, then disables expression evaluator dereferencing, and re-evaluates the expression:

```
SI>SET EE_IMPL_DEREF on
EE_IMPL_DEREF = on
SI>? a
=>> 00000007 (7) "." 0000 0111
int
SI>? &a
=>> 0012fed4 (1244884) "Ôþ." 0000 0000 0001 0010 1111 1110 1101 0100
int*
SI>SET EE_IMPL_DEREF off
EE_IMPL_DEREF = off
SI>? a
=>> 0012fed4 (1244884) "Ôþ." 0000 0000 0001 0010 1111 1110 1101 0100
int
SI>? *a
=>> 00000007 (7) "." 0000 0111
int
```

See Also

?, EVAL, SET

SET EXEPATH

Set or add a search path for image files.

Syntax

```
SET EXEPATH [-a] search-path
```

-a	Append an image source search path to the current search path.
search-path	The image source search path.

Use

Use the SET EXEPATH command to set or add an image source search path. Using the *-a* parameter appends a new image path to the existing path. If you do not use the *-a* parameter, SET EXEPATH will override the previous path. SET EXEPATH only accepts valid (existing) paths. Trying to set EXEPATH to an invalid path value results in the attempt being ignored, and the value of EXEPATH is reset to null, unless you used the *-a* parameter to preserve the previous setting.

Using ellipses (...) in the path name indicates all subdirectories of the specified directory path. For example, `C:\IMAGES\...` indicates all subdirectories of `C:\IMAGES`.

To view the paths that are currently set for the image source, use the SET EXEPATH command without any parameters.

Example

The following example sets `c:\myimages` as the image source path:

```
SI>SET EXEPATH c:\myimages
EXEPATH = c:\myimages
```

The following example adds `c:\winnt` to the image source path:

```
SI>SET EXEPATH -a c:\winnt
EXEPATH = c:\myimages;c:\winnt
```

See Also

SET, SET EXPORTPATH, SET KDEXTPATH, SET SCRIPTPATH, SET SRCPATH, SET SYMPATH

SET EXPORTPATH

Set or add a file search path to the location of exports.

Syntax

```
SET EXPORTPATH search-path
```

search-path	The file system search path.

Use

The SET EXPORTPATH command sets a file system search path for exports. SET EXPORTPATH only accepts valid (existing) paths. Trying to set EXPORTPATH to an invalid path value results in the attempt being ignored, and the value of EXPORTPATH is reset to null.

Using ellipses (...) in the path name indicates all subdirectories of the specified directory path. For example, `C:\EXPORTS\...` indicates all subdirectories of `C:\EXPORTS`. Any directory named by SET EXPORTPATH must exist, or the command will fail.

Use SET EXPORTPATH in conjunction with the ADDEXP command to set a destination directory on the master for a local cache of export information extracted from the target. After setting the export path, issue the ADDEXP command to retrieve exports from the target and place them in the local cache. Once exports are stored in the local cache, Visual SoftICE will automatically load them anytime symbols are not found.

To view the path that is currently set for exports, use the SET EXPORTPATH command without any parameters.

Examples

The following example sets `c:\myexports` as the file system search path for exports:

```
SI>SET EXPORTPATH c:\myexports
EXPORTPATH = c:\myexports
```

The following example sets `c:\myexports` and all its subdirectories as the file system search path for exports:

```
SI>SET EXPORTPATH c:\myexports\...
EXPORTPATH = c:\myexports...
```

See Also

GETEXP, SET, SET EXEPATH, SET KDEXTPATH, SET SCRIPTPATH, SET SRCPATH, SET SYMPATH

SET FLOATREGFORMAT

Enable or disable formatting floating-point registers.

Syntax

```
SET FLOATREGFORMAT [on | off]
```

on	Enable formatting FP registers.
off	Disable formatting FP registers.

Use

Use the SET FLOATREGFORMAT to enable or disable the formatting of FP registers.

To view the current setting for the floating point register format, use the SET FLOATREGFORMAT command without any parameters.

Example

The following example enables formatting FP registers:

```
SI>SET FLOATREGFORMAT on
FLOATREGFORMAT = on
```

See Also

R, SET, SET REGNAME

SET GLOBALBREAK

Set the mode for image breakpoints.

Syntax

```
SET GLOBALBREAK [load | off]
```

load Set the mode to break on loading an image file.

off Disable the option.

Use

Use the SET GLOBALBREAK command to control if Visual SoftICE breaks on loading of image files. Selecting *off* disables the option.

To view the current GLOBALBREAK setting, use the SET GLOBALBREAK command without any parameters.

Example

The following example configures Visual SoftICE to break when an image loads:

```
SI>SET GLOBALBREAK load
GLOBALBREAK = Load
```

See Also

BC, BD, BE, BL, BMSG, BPINT, BPIO, BPLOAD, BPM, BPR, BPX, BSTAT, EXEC, KILL, SET, SVCSTART, SVCSTOP

SET IMAGEMATCH

Configure the way the symbol engine and the target attempt to match symbolic data to the actual running image (module) on the target.

Syntax

```
SET IMAGEMATCH [exact | best]
```

exact Symbol data will only be used if it exactly matches the image in question.

best Symbol data will be used if it is a close match to the image in question.

Use

Use the SET IMAGEMATCH command to configure the way the symbol engine and the target attempt to match symbolic data (.dbg, .pdb, or image header information) to the actual running image (module) on the target. Selecting *exact* matching means that symbol data will only be used if it exactly matches the image in question. Selecting *best* matching allows for the symbol data to be used if it is a close match to the image in question. Since there are times that useful symbols are available but might differ from the target image by timestamp or version, the *best* matching setting is the default setting.

To view the current IMAGEMATCH setting, use the SET IMAGEMATCH command without any parameters.

Example

The following example configures image matching for exact matches only:

```
SI>SET IMAGEMATCH exact
IMAGEMATCH = Exact
```

The following example configures image matching for the best match:

```
SI>SET IMAGEMATCH best
IMAGEMATCH = Best
```

See Also

SET

SET KDEXTPATH

Set or add a file search path to find KD extensions.

Syntax

```
SET KDEXTPATH [-a] search-path
```

-a	Append a search path to the current KD extension path.
search-path	The file system search path.

Use

Use the SET KDEXTPATH command to set a file system search path for KD extensions. Using the *-a* parameter appends a new KD extension path to the existing path. If you do not use the *-a* parameter, SET KDEXTPATH will override the previous path. SET KDEXTPATH only accepts valid (existing) paths. Trying to set KDEXTPATH to an invalid path value results in the attempt being ignored, and the value of KDEXTPATH is reset to null, unless you used the *-a* parameter to preserve the previous setting.

Using ellipses (...) in the path name indicates all subdirectories of the specified directory path. For example, `C:\KDEXT\...` indicates all subdirectories of `C:\KDEXT`.

To view the paths that are currently set for KD extensions, use the SET KDEXTPATH command without any parameters.

Examples

The following example sets `c:\mykdext` as the file system search path for KD extensions:

```
SI>SET KDEXTPATH c:\mykdext
KDEXTPATH = c:\mykdext
```

The following example sets c:\mykdext and all its subdirectories as the file system search path for KD extensions:

```
SI>SET KDEXTPATH c:\mykdext\...
KDEXTPATH = c:\mykdext...
```

See Also

SET, SET EXEPATH, SET EXPORTPATH, SET SCRIPTPATH, SET SRCPATH, SET SYMPATH

SET MSGLEVEL

Set the message filtering level to display the target and execution messages.

Syntax

```
SET MSGLEVEL [on | off | verbose]
```

on Show breakpoint, stepping, and critical event messages. This is the default setting.

off Only show critical event messages.

verbose Show all events.

Use

Use the SET MSGLEVEL command to control the target and execution messages. Turning on the event messaging with either `SET MSGLEVEL on` or `SET MSGLEVEL verbose` displays event messages from the target in the window where you executed the SET MSGLEVEL command.

A critical event is an event requiring your immediate attention, such as a fault, bugcheck, or shutdown.

The default state for event messages is *on*.

To view the current MSGLEVEL setting, use the SET MSGLEVEL command without any parameters.

Example

The following example reports only critical event messages from the target:

```
SI>SET MSGLEVEL off
MSGLEVEL = off
```

See Also

SET

SET PACKETFORMAT

Set the format of the PACKET command.

Syntax

```
SET PACKETFORMAT [RAW_LINE | RAW_DETAIL | STANDARD_LINE |
STANDARD_DETAIL | STRUCTURE]
```

RAW_LINE	Displays one hexadecimal line per packet.
RAW_DETAIL	Displays detailed hexadecimal packet information.
STANDARD_LINE	Displays one formatted/interpreted line per packet.
STANDARD_DETAIL	Displays detailed formatted/interpreted packet information.
STRUCTURE	Produces a structured element dump.

Use

Use the SET PACKETFORMAT command to set the format of the PACKET command. The default packet format is STANDARD_DETAIL.

To view the current PACKETFORMAT setting, use the SET PACKETFORMAT command without any parameters.

Example

The following example shows the SET PACKETFORMAT command setting the format to RAW_DETAIL.

```
SI>SET PACKETFORMAT RAW_DETAIL
PACKETFORMAT = RAW_DETAIL
```

See Also

PACKET, SET

SET RADIX

Set the radix of data for input and output.

Note: Only decimal and hexadecimal are currently supported.

Syntax

```
SET RADIX [DEC | HEX]
```

DEC	Enable decimal data.
HEX	Enable hexadecimal data.

Use

Use the SET RADIX command to control the input interpretation and output formatting of anything other than addresses, which are always hexadecimal.

To view the current RADIX setting, use the SET RADIX command without any parameters.

Example

The following example sets the radix of data to hexadecimal format:

```
SI>SET RADIX HEX
RADIX = HEX
```

See Also

SET

SET REGNAME

Select a register name set.

Syntax

```
SET REGNAME [asm | hw | os]
```

asm Common assembler/disassembler register names.

hw Hardware (defined by CPU manufacturer) register names.

os Operating system register names.

Use

Use the SET REGNAME to select a register name set to display. ASM is the default setting.

To view the current REGNAME setting, use the SET REGNAME command without any parameters.

Example

The following example selects operating system register names:

```
SI>SET REGNAME os
REGNAME = OS (register names as used by the OS)
```

See Also

R, RG, SET, Understanding Register Names and Groups in the on-line help

SET SCRIPTECHO

Echo script commands to the current Command page.

Syntax

```
SET SCRIPTECHO [on | off]
```

on Enable echoing script commands to the console (default).

off Disable echoing script commands to the console.

Use

Use the SET SCRIPTECHO command to configure the target to echo script commands to the current console.

To view the current SCRIPTECHO setting, use the SET SCRIPTECHO command without any parameters.

Example

The following example enables script echoing:

```
SI>SET SCRIPTECHO on
SCRIPTECHO = on
```

The following example disables script echoing:

```
SI>SET SCRIPTECHO off
SCRIPTECHO = off
```

See Also

SET

SET SCRIPTPATH

Set a file system search path for scripts.

Syntax

```
SET SCRIPTPATH [-a] search-path
```

-a	Append a search path to the current script path.
search-path	The file system search path.

Use

Use the SET SCRIPTPATH command to set a file system search path for scripts. If you have set a scriptpath and enter a script filename that cannot be found or has no path, Visual SoftICE will search in the path designated by *filespec* for a file of the same name and attempt to execute that script. SET SCRIPTPATH only accepts valid (existing) paths. Trying to set SCRIPTPATH to an invalid path value results in the attempt being ignored, and the value of SCRIPTPATH is reset to null, unless you used the *-a* parameter to preserve the previous setting.

Using ellipses (...) in the path name indicates all subdirectories of the specified directory path. For example, `C:\SCRIPTS\...` indicates all subdirectories of `C:\SCRIPTS`.

To view the paths that are currently set for scripts, use the SET SCRIPTPATH command without any parameters.

Examples

The following example sets `c:\ntscripts` as the file system search path for scripts:

```
SI>SET SCRIPTPATH c:\ntscripts
SCRIPTPATH = c:\ntscripts
```

The following example sets `c:\myscripts` and all its subdirectories as the file system search path for scripts:

```
SI>SET SCRIPTPATH c:\myscripts\...
SCRIPTPATH = c:\myscripts...
```

See Also

SET, SET EXEPATH, SET EXPORTPATH, SET KDEXTPATH, SET SRCPATH, SET SYMPATH

SET SCRIPTSTOPONERROR

Controls whether scripts automatically stop execution when an error occurs.

Syntax

```
SET SCRIPTSTOPONERROR [on | off]
```

on Enables the option (default).

off Disables the option.

Use

Use the SET SCRIPTSTOPONERROR command to control whether scripts will automatically stop execution when an error occurs. The default for this option is *on*.

To view the current SCRIPTSTOPONERROR setting, use the SET SCRIPTSTOPONERROR command without any parameters.

Example

The following example disables the option:

```
SI>SET SCRIPTSTOPONERROR off
SCRIPTSTOPONERROR = off
```

See Also

SET

SET SRCPATH

Set a search path for source files.

Syntax

```
SET SRCPATH [-a] search-path
```

-a Append a search path to the current source file path.

search-path The source file search path.

Use

Use the SET SRCPATH command to set a source file search path. SET SRCPATH only accepts valid (existing) paths. Trying to set SRCPATH to an invalid path value results in the attempt being ignored, and the value of SRCPATH is reset to null, unless you passed the *-a* parameter to preserve the previous setting.

Using ellipses (...) in the path name indicates all subdirectories of the specified directory path. For example, `C:\SOURCE\...` indicates all subdirectories of `C:\SOURCE`.

To view the paths that are currently set for the source, use the SET SRCPATH command without any parameters.

Examples

The following example sets `c:\mysource` as the source file search path:

```
SI>SET SRCPATH c:\mysource
SRCPATH = c:\mysource
```

See Also

SET, SET EXEPATH, SET EXPORTPATH, SET KDEXTPATH, SET SCRIPTPATH, SET SYMPATH

SET STEPMODE

Set the stepping mode for function keys.

Syntax

```
SET STEPMODE [src | instr | wnd]
```

src	Sets Source stepping mode.
instr	Sets Instruction stepping mode.
wnd	Sets current Window stepping mode.

Use

Use the SET STEPMODE command to set the stepping mode for function keys. The stepping mode controls what type of stepping the function keys **F10**, **F11**, and **Shift+F11** execute when pressed.

If the current mode is INSTRUCTION, then pressing **F10** issues an Instruction step command (P), regardless of which page has focus. Pressing **F11** or **F8** causes an Instruction Step-Into command (T) to be issued. Pressing **Shift+F11** causes an Instruction Step-Out Of command (P ret) to be issued.

If the current mode is SOURCE, then pressing **F10** issues a Source step command, regardless of which page has focus. Pressing **F11** or **F8** causes a Source Step-Into command to be issued. Pressing **Shift+F11** causes a Source Step-Out Of command to be issued.

Note: There are no command line commands to specifically source step.

The WND mode issues a command to the current or last activated code page (Source or Disassembly) to execute the requested step. If the current page (the page that has focus) is not a code page, Visual SoftICE issues the command to the last activated code page. If the last activated code page is a disassembly view, it always issues instruction step commands as it would in Instruction Stepping mode. If the last activated page is a source view, Visual SoftICE take appropriate action based on the current Instruction Pointer (IP).

To view the current STEPMODE setting, use the SET STEPMODE command without any parameters.

Example

The following example sets the stepping mode as src.

```
SI>SET STEPMODE src
STEPMODE = SRC - Source Stepping
```

See Also

SET

SET STICKYCONTEXT

Control auto-process context selection.

Syntax

```
SET STICKYCONTEXT [on | off]
```

on Enable maintaining the current process context.

off Allow auto-process context selection.

Use

Use the SET STICKYCONTEXT command to indicate whether you want the user interface to maintain the current process context, or allow Visual SoftICE to automatically change process contexts.

When STICKYCONTEXT is turned *on*, issuing a stop command (STOP, CTRL-D. or the toolbar button) will cause the UI to auto-ADDR to the context that was active when you enabled STICKYCONTEXT. If that context no longer exists, the STICKYCONTEXT mode is reset to *off*. Any other stop event, such as a breakpoint or fault, will not cause the UI to auto-ADDR.

To view the current STICKYCONTEXT setting, use the SET STICKYCONTEXT command without any parameters.

Example

The following example enables maintaining the current process context:

```
SI>SET STICKYCONTEXT on
STICKYCONTEXT = on
```

The following example disables auto-process context selection:

```
SI>SET STICKYCONTEXT off
STICKYCONTEXT = off
```

See Also

SET

SET STOPONCMD

Stop the target when you issue any console command.

Syntax

```
SET STOPONCMD [on | off]
```

on	Enables the option.
off	Disables the option.

Use

Use the SET STOPONCMD command to stop the target whenever you issue a command that would be sent to the target. The target stops before it executes the command sent to it.

The target will not be restarted automatically. You must restart the target directly by issuing the GO command.

To view the current STOPONCMD setting, use the SET STOPONCMD command without any parameters.

Example

The following example enables STOPONCMD:

```
SI>SET STOPONCMD on
STOPONCMD = on
```

See Also

GO, SET

SET SYMPATH

Set or add a search path for symbol files.

Syntax

```
SET SYMPATH [-a] search-path
```

-a Append a search path to the current symbol search path.

search-path The symbol search path.

Use

Use the SET SYMPATH command to set or add a symbol search path. Using the *-a* parameter appends a new symbol search path to the existing path. If you do not use the *-a* parameter, SET SYMPATH overrides the existing path. SET SYMPATH only accepts valid (existing) paths. Trying to set SYMPATH to an invalid path value results in the attempt being ignored, and the value of SYMPATH is reset to null, unless you used the *-a* parameter to preserve the previous setting.

Using ellipses (...) in the path name indicates all subdirectories of the specified directory path. For example, `C:\SYMBOLS\...` indicates all subdirectories of `C:\SYMBOLS`.

Adding a Symbol Server to your symbol path allows Visual SoftICE to retrieve symbols from that server if they cannot be found locally. To add a symbol server to your path, the generic format of the command is as follows:

```
SET SYMPATH -a "srv*LocalCache*\\Server\Share"
```

To specifically add the Microsoft Symbol Server to your path, execute the following command:

```
SET SYMPATH -a "srv*c:\symserver*http://msdl.microsoft.com/
download/symbols"
```

To view the paths that are currently set for the symbol source, use the SET SYMPATH command without any parameters.

Examples

The following example sets `c:\mysymbols` as the symbol search path:

```
SI>SET SYMPATH c:\mysymbols
SYMPATH = c:\mysymbols
```

The following example adds c:\winnt to the symbol search path:

```
SI>SET SYMPATH -a c:\winnt
SYMPATH = c:\mysymbols;c:\winnt
```

See Also

ADDSYM, SET, SET EXEPATH, SET EXPORTPATH, SET KDEXTPATH, SET SCRIPTPATH, SET SRCPATH, SET SYMSRVSEARCH, TABLE

SET SYMSRVSEARCH

Control symbol server searching behavior.

Syntax

```
SET SYMSRVSEARCH on | off
```

on Enables searching any symbol server identified in the system symbol path (default).

off Disables symbol server searching, even if you have servers identified in the symbol search path.

Use

Use the SET SYMSRVSEARCH command to control the searching for symbols through symbol servers. The default value for SYMSRVSEARCH is *on*, where any symbol server searches are enabled.

If you set the value of SYMSRVSEARCH to *off*, symbol servers will not be searched, even if they are included in the symbol search path.

To view the current value of SYMSRVSEARCH, use the SET SYMSRVSEARCH command without any parameters.

Examples

The following example disables symbol server searching:

```
SI>SET SYMSRVSEARCH off
SYMSRVSEARCH = off
```

The following example enables symbol server searching:

```
SI>SET SYMSRVSEARCH on
SYMSRVSEARCH = on
```

See Also

ADDSYM, DELSYM, LOAD, RELOAD, SET, SET SYMPATH, SYM, UNLOAD

SET SYMTABLEAUTOLOAD

Control symbol auto-load behavior.

Syntax

```
SET SYMTABLEAUTOLOAD [on | off]
```

on Turn auto-loading of symbols on (default).

off Turn auto-loading of symbols off.

Use

Use the SET SYMTABLEAUTOLOAD command to control the on-demand automatic loading of symbols by the symbol engine. The default value for SYMTABLEAUTOLOAD is *on*, where the symbol engine automatically loads symbols on-demand. In this mode, if you specifically unload a symbol table, that table is marked not to auto-load again.

If you set the value of SYMTABLEAUTOLOAD to *off*, automatic symbol loading is disabled. If you unload a symbol table, it will not be marked, and may auto-load again if auto-loading is re-enabled.

To view the current value of SYMTABLEAUTOLOAD, use the SET SYMTABLEAUTOLOAD command without any parameters.

Examples

The following example disables automatic symbol loading:

```
SI>SET SYMTABLEAUTOLOAD off
SYMTABLEAUTOLOAD = off
```

The following example enables automatic symbol loading:

```
SI>SET SYMTABLEAUTOLOAD on
SYMTABLEAUTOLOAD = on
```

See Also

ADDSYM, LOAD, RELOAD, SET, TABLE, UNLOAD

SET THREADP

Control thread-specific stepping.

Syntax

```
SET THREADP [on | off]
```

on	Enable thread-specific stepping (default).
off	Disable thread-specific stepping.

Use

Use the SET THREADP command to configure stepping on the target. When disabled, stepping is not constrained to the current thread.

Stepping in Visual SoftICE is thread-specific by default. If the current instruction pointer is executing in thread X, Visual SoftICE will not break until the program step occurs in thread X. This prevents the case of the operating system 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 view the current setting for target thread stepping, use the SET THREADP command without any parameters.

Example

The following example enables thread-specific stepping:

```
SI>SET THREADP on
THREADP = on
```

The following example disables thread-specific stepping:

```
SI>SET THREADP off
THREADP = off
```

See Also

SET

SET UIQ_THRESHOLD

Set the user interface's responsiveness by managing how many events can queue up unprocessed from the target.

Syntax

```
SET UIQ_THRESHOLD [n]
```

n The number of events, in hex, that the UI allows to queue up unprocessed from the target before it halts the target to catch up. The default setting is 0x00000019 (25 decimal).

Use

Use the SET UIQ_THRESHOLD command to control the responsiveness of the UI in situations where a target is generating a large number of events. The number you set (in hex) corresponds to the number of events that the UI will allow to queue up, before it halts the target and catches up with the processing of the queued events. Assigning a value of `0xFFFFFFFF` (-1 decimal) disables the feature.

To view the current UIQ_THRESHOLD setting, use the SET UIQ_THRESHOLD command without any parameters.

Example

The following example sets the threshold to `f0` (240 decimal):

```
SI>SET UIQ_THRESHOLD f0
UIQ_THRESHOLD = f0 (240)
```

See Also

SET

SET UPPERCASE

Set uppercase hexadecimal disassembly output.

Syntax

```
SET UPPERCASE [on | off]
```

on	Enable uppercase hex disassembly output.
off	Disable uppercase hex disassembly output.

Use

Use the SET UPPERCASE command to set uppercase hex disassembly output, such that all hex values returned by the target will be displayed in uppercase (for instance, `0x4af` will always be displayed as `0x4AF`).

To view the current UPPERCASE setting, use the SET UPPERCASE command without any parameters.

Example

The following example sets uppercase hex disassembly output:

```
SI>SET UPPERCASE on
UPPERCASE = on
```

See Also

SET

SET WARNLEVEL

Set warning and confirmation level.

Syntax

```
SET WARNLEVEL [high | low | off]
```

high Warn of any action that could be harmful or fatal to the target.

low Warn only on severe actions that could be fatal to the target.

off Do not warn or confirm any actions.

Use

Use the SET WARNLEVEL to set the warning and confirmation level. Low is the default setting.

To view the current WARNLEVEL setting, use the SET WARNLEVEL command without any parameters.

Example

The following example sets the warning level to high:

```
SI>SET WARNLEVEL high
WARNLEVEL = High
```

See Also

SET

SHUTDOWN

Shut down the target machine you are currently connected to.

Note: The target must be running for this command to succeed.

Syntax

```
SHUTDOWN
```

Use

Use the SHUTDOWN command to shut down the currently connected target machine.

Example

The following example shuts down the target machine:

```
SI>SHUTDOWN
```

See Also

HBOOT, REBOOT, STOP

SLEEP

Halt execution on the master for a specified number of milliseconds.

Syntax

```
SLEEP msecs (in decimal)
```

msecs Number of milliseconds (in decimal) to halt execution on the master.

Use

SLEEP halts execution on the master for a specified number of milliseconds. This command is used mainly in script execution.

Example

The following example is a simple script that uses SLEEP:

```
con 172.23.100.158
sleep 5
sympath -a p:\pablo\fib64\debug_ia64
exepath -a p:\pablo\fib64\debug_ia64
sympath -a p:\pablo\fib64\fib1\debug_ia64
exepath -a p:\pablo\fib64\fib1\debug_ia64
addr fib64_2
table fib1.dll
table
// script c:\dev\source\gui\fib64.txt
```

See Also

@, SCRIPT

SS

Search all open files for specified string.

Syntax

```
SS [line-number] 'string'
```

line-number	Starting line number for search.
string	String to search for.

Use

Use the SS command to search all the open files for a specified string. If you specify a line number, Visual SoftICE starts the search at that line. If you do not specify a line number, then the search will start at the current top line.

Example

The following example searches all the open files for FOOBAR starting at line number 42.

```
SI>SS 42 'FOOBAR'
```

See Also

FILE, FS

STACK

Display a call stack.

Syntax

```
STACK -r -d [-t trap_frame] [thread-ID]
```

-r	Force a refresh of the call stack.
-d	Display detailed information for the stack frame.
-t	Start stack-walking from a trap-frame address.
trap_frame	Address of a trap frame.
thread-ID	The ID of the thread.

Use

Use the STACK command to display the call stack for a thread.

If you enter STACK with no parameters, the stack frames for the current process are displayed. You can explicitly specify a different thread or process via a thread identifier.

If you use the -*t* option and specify an address with the *stack_frame* parameter, Visual SoftICE begins stack-walking from the specified trap frame address within the current thread context.

In a situation where you have loaded new tables, for example using the ADDSYM command, you may want to force a refresh of the call stack. To force a refresh of the call stack, use the -*r* option.

To show detailed information for a stack frame, including the function table (FPO or Unwind data) and parameters, use the -*d* option.

The STACK command walks the stack starting at the base by traversing target stack frames. If an invalid stack frame or address that has been paged out is encountered during the walk, the traversal will stop. Visual SoftICE will continue walking the stack through ring transitions. Walking the IA64 target stack requires access to unwind descriptors located in the image file. Normally these are read from memory at the target machine, but if the required unwind descriptors are paged out, the stack walking will stop. You can guarantee access to the unwind descriptors by making a copy of the image available on the master in one of the directories referenced by the image search path (refer to SET EXEPATH).

Note: If the current address context is not set to the context of the specified TID, the command will fail with a "Specified Thread Not Found" error.

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.

The STACK command output includes the stack pointer, the instruction pointer, and the symbol information 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 stack status column provides extra information about the stack walking process. It notifies you about potential problems Visual SoftICE encountered while walking the stack. If Visual SoftICE encountered no problems, and the stack was walked fully, no status messages are displayed. If the stack walk terminates prematurely there will be a status message describing the terminating condition. The most important message is the "Unwind info unavailable" message on IA64.

"Unwind info unavailable" usually means that Visual SoftICE could not read the unwind information from the target. This occurs because the part of the image on the target containing the unwind information is paged out and Visual SoftICE could not access it. A solution for this problem is to have a local copy of all executables (including files in the system32 directory). Add these directories to the exepath so the local copies can be found instead of having to be retrieved from the target.

This situation can also occur in some circumstances after you have set your EXEPATH. If this happens, reload the table. Refer to the Troubleshooting section of the on-line help for more information on getting unwind information when stack-walking IA64/AMD64 machines.

For IA64 and AMD64, unwind information is displayed in the Function Table column using the format UNWIND(*offset*), where *offset* is the offset from the image base address to the location of the unwind data.

For x86, the Function Table column displays FPO data. The FPO data displayed is, in order: the **Frame Type**, **Size of Parameters in DWORDs**, **Number of Registers Saved**, **Number of Local Variables**, and whether the **EBP Register has been Allocated**.

The call stack support is not limited to applications; it will also work for device drivers.

Output

The detailed output for stack frames contains the following information:

- Context
- Instruction Pointer
- Stack Pointer
- Frame Pointer
- Parameters
- Function Table
- Status

Example

The following example uses the *-t* option with the address `f1d04d64` to begin stack walking within the current thread context.

```
SI>STACK -t f1d04d64
Context                                Instruction Ptr  Stack Ptr  Frame Ptr  Status
------------------------------------------------------------------------------------
ntoskrnl!KeBugCheckEx+19               804fc1bb         f1d04d64   f1d04d64
SystemCallStub+4 (Ring 3/Ring0 Trans.)  7ffe0304         0012fb18   0012fb18
ntdll!NtDeviceIoControlFile+c          77f7e7df         0012fb20   0012fb7c
TEST32!_Section.text+46ed              004056ed         0012fb84   0012fc10
TEST32!_Section.text+40b0              004050b0         0012fc18   0012fdac
USER32!InternalCallWinProc+1b          77d43a5f         0012fdb4   0012fdd8
USER32!UserCallWinProcCheckWow+b7      77d43b2e         0012fde0   0012fe40
USER32!DispatchMessageWorker+dc        77d43d6a         0012fe48   0012fea0
USER32!DispatchMessageA+b              77d441fd         0012fea8   0012feac
TEST32!_Section.text+378a              0040478a         0012feb4   0012ff20
TEST32!_Section.text+4e83              00405e83         0012ff28   0012ffc0
kernel32!BaseProcessStart+23           77e7eb69         0012ffc8   0012fff0
```

The following example passes a thread ID of `7bc` and uses the *-t* option with the address `f1c9cb98` to begin stack walking.

```
SI>STACK -t f1c9cb98 7bc
Context                                Instruction Ptr  Stack Ptr  Frame Ptr  Status
-------------------------------------------------------------------------------------
ntoskrnl!KeBugCheckEx+19               804fc1bb         f1c9cb98   f1c9cb98
Testdrv!TestdrvIoctl+8c                f976d73c         f1c9cc0c   f1c9cc34
ntoskrnl!IopfCallDriver+27             804ec04f         f1c9cc3c   f1c9cc58
...
TEST32!WinMain+9a                      0040478a         0012feb4   0012ff20
TEST32!WinMainCRTStartup+1b3           00405e83         0012ff28   0012ffc0
kernel32!BaseProcessStart+23           77e7eb69         0012ffc8   0012fff0
```

The following example uses the *-d* option on x86.

Note: The actual output of the following example contains more information than is displayed. The example has been edited and truncated in order to better fit in the manual.

```
SI>stack -d
Context                                Instruction Ptr    Stack Ptr    Frame Ptr    Parameters
------------------------------------------------------------------------------------------------
ntoskrnl!KeBugCheckEx+19               804fc1bb           f1d04b64     f1d04b7c     7f,d,0,0,0,
ntoskrnl!KiSystemFatalException+e      804d8362           f1d04b84     f1d04b98
Testdrv!TestdrvIoctl+a2                f9801752           f1d04c0c     f1d04c34
ntoskrnl!IopfCallDriver+27             804ec04f           f1d04c3c     f1d04c44
ntoskrnl!IopSynchronousServiceTail+58  80571c0a           f1d04c4c     f1d04c58     ffa2a2a8,
ntoskrnl!IopXxxControlFile+29f         80571f1e           f1d04c60     f1d04d00     7bc,0,0,0,
ntoskrnl!NtDeviceIoControlFile+28      805863d5           f1d04d08     f1d04d34     7bc,0,0,0,
ntoskrnl!KiSystemService+c4            804d4e91           f1d04d3c     f1d04d64
SystemCallStub+4 (Ring 3/Ring0 Trans.)  7ffe0304           0012fb18     0012fb18
ntdll!NtDeviceIoControlFile+c          77f7e7df           0012fb20     0012fb1c     7bc,0,0,0,
kernel32!DeviceIoControl+286           77e73fcb           0012fb24     0012fb7c     7bc,9c416004,
TEST32!_Section.text+46ed              004056ed           0012fb84     0012fc10     c0150,1f5,
TEST32!_Section.text+40b0              004050b0           0012fc18     0012fdac     c0150,111,1f5,
USER32!InternalCallWinProc+1b          77d43a5f           0012fdb4     0012fdd8     4010af,c0150,
USER32!UserCallWinProcCheckWow+b7      77d43b2e           0012fde0     0012fe40     0,4010af,
USER32!DispatchMessageWorker+dc        77d43d6a           0012fe48     0012fea0     12ff04,1,
USER32!DispatchMessageA+b              77d441fd           0012fea8     0012feac     12ff04,
TEST32!_Section.text+378a              0040478a           0012feb4     0012ff20     400000,0,
TEST32!_Section.text+4e83              00405e83           0012ff28     0012ffc0     44005c,470042,
kernel32!BaseProcessStart+23           77e7eb69           0012ffc8     0012fff0     405cd0,
```

See Also

SET EXEPATH, THREAD

STARTDEBUGGER

Start a Visual SoftICE debugger on the currently connected target.

Note: The target must be running for this command to succeed.

Syntax

```
STARTDEBUGGER
```

Use

Use the STARTDEBUGGER command to start a Visual SoftICE debugger on the currently connected target machine.

Example

The following example starts a Visual SoftICE debugger on the target machine:

```
SI>STARTDEBUGGER
```

STOP

Stop the target you are currently connected to.

Syntax

```
STOP
```

Use

Use the STOP command to stop the currently connected target.

While stepping, you can abort the active step by issuing the STOP command, clicking **Stop** on the toolbar, or pressing **Ctrl-Break**. If the step was issued from the command page, then the red **Abort Command** button will abort it as well.

Note: Whether or not the target responds to the stop that the master issues depends on the state of the target.

Example

The following example stops the target:

```
SI>STOP
```

See Also

HBOOT, REBOOT, SHUTDOWN

SVCSTART

Start a service on the target machine you are currently connected to.

Note: The target must be running for this command to succeed.

Syntax

```
SVCSTART [service-name]
```

service-name	Name of the service you want to start on the target.

Use

Use the SVCSTART command to start a service on the currently connected target machine.

Example

The following example starts myservice on the target machine:

```
SI>SVCSTART myservice
```

See Also

EXEC, KILL, SET GLOBALBREAK, SVCSTOP

SVCSTOP

Stop a service on the target machine you are currently connected to.

Note: The target must be running for this command to succeed.

Syntax

```
SVCSTOP [service-name]
```

service-name Name of the service you want to stop on the target.

Use

Use the SVCSTOP command to stop a service on the currently connected target machine.

Example

The following example stops myservice on the target machine:

```
SI>SVCSTOP myservice
```

See Also

EXEC, KILL, SET GLOBALBREAK, SVCSTART

SYM

Display or set a symbol.

Syntax

```
SYM -v [image!]symbol-name
```

-v	Verbose. Show the symbol with full type information instead of just the name.
image!	Optional image name to specify the symbol table to search.
symbol-name	A valid symbol-name. Wildcard characters are fully supported.

Use

Use the SYM command to display and set symbols with global addresses, such as functions, global data, and exported public symbols. If you enter SYM without parameters the default symbol-name is asterisk (*), and all symbols in the current symbol table are displayed. Visual SoftICE displays the address and category of each symbol next to the symbol-name.

If you specify a symbol-name without a value, the symbol-name and its address are displayed. If the symbol-name is not found, nothing is displayed.

If you use *-v* with the SYM command you specify verbose mode. Visual SoftICE will display as much type information as is available. For instance, a function symbol will be displayed with the return type and parameter types.

The SYM command is often useful for finding a symbol when you can only remember a portion of the name.

Output

These categories are only shown if debug information is available.	**Function**	A fully-typed Function symbol.
	Thunk	A small function generated by the compiler to call another function external to the image.
	Data	A fully-typed Data symbol.
	Public	An exported symbol (Function or Data). Public symbols have no type information.
These categories are un-typed names generated by Visual SoftICE from information available in the image.	**Exports**	Exports are stored in the image file. These are the same as Public symbols in the debug information. Items exported by ordinal numbers that have no names are shown with a generated name of the form `_Ordinal<n>` where <n> is the decimal ordinal number.
	Section	Section names show in the form `_Section<name>` where <name> is the section name. Many section names begin with a period (.) so a typical section name is `_Section.text`.
	Image	Miscellaneous image names. `_Base` is the base address of the image. `_EntryPoint` is the image entry point (it may be zero if there is no entry point). `_FunctionAtRva<x>` where <x> is the relative virtual address of the function in the image (IA64 only). The `_FunctionAtRva<x>` names are generated from the function table in the IA64 `.pdata` section.

If you specify a value, the address of all symbols that match symbol-name are set to the value.

Example

The following example displays all symbols in section `fib64_2` that exactly match `main`:

```
SI>SYM fib64_2!main
fib64_2!00020e20 Function  main
fib64_2!001a2fa8 Public    main
```

The following example displays all symbols in section fib64_2 that exactly match fib_func, using the *-v* parameter to give verbose information:

```
SI>SYM -v fib64_2!fib_func
0000000000424440 Function  int fib_func(float*,float*,float*)
00000000005a32f0 Public    fib_func
```

The following example displays all symbols in section fib64_2 that match with some pattern of main, using the * wildcard:

```
SI>SYM fib64_2!*main*
fib64_2!00020e20 Function  main
...
fib64_2!000cc560 Public    .mainCRTStartup
```

The following example displays all sections in image fib64_2:

```
SI>SYM fib64_2!_Section*
0000000000402000 Section   _Section.text
...
00000000005e0000 Section   _Section.idata
```

The following example locates the first image that contains a symbol that starts with fib and shows all symbols that start with fib in that image. This form might take a long time because Visual SoftICE must load every symbol table until an instance of the symbol is found.

```
SI>SYM *!fib*
0000000000424440 Function  fib_func
...
00000000005a3cb0 Public    fibulation
```

Note: When no explicit image name is specified, the SYM command searches the current symbol table first, and then searches all other symbols tables that are already loaded.

See Also

EXP, GETEXP, SET SYMPATH, SET SYMSRVSEARCH, TABLE

T

Trace one instruction.

Syntax

```
T [=start-address] [count]
```

start-address	Address at which to begin execution.
count	Specify how many times Visual 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 Instruction Pointer (IP), unless you specify the *start-address* parameter. If you specify this parameter, the current IP is changed to the *start-address* prior to single-stepping. If you attempt to set a *start-address* that is outside the current function scope and the warning level is not set to *off*, then Visual SoftICE generates a warning message asking you to confirm the new *start-address*.

If you specify *count*, Visual SoftICE single-steps *count* times.

If a Register page is visible when the target stops, all registers that were altered since the T command was issued are highlighted.

While stepping, you can abort the active step by issuing the STOP command, clicking **Stop** on the toolbar, or pressing **Ctrl-Break**. If the step was issued from the command page, then the red **Abort Command** button will abort it as well.

Note: Whether or not the target responds to the stop that the master issues depends on the state of the target.

Example

The following example single-steps through three instructions starting at the memory location equal to the current value of the system instruction pointer plus 20 bytes on an IA-64 target:

```
SI>T = ip + 20 3
```

See Also

P, STOP

TABLE

Change or display the current symbol table.

Syntax

```
TABLE [-v] [-u] [-s] [table-name]
```

-v	Verbose mode. Shows all table regardless of whether symbols are currently loaded.
-u	User mode. Shows user-mode (Ring 3) tables only.
-s	System mode. Shows system (Ring 0) tables only.
table-name	A valid table-name. Wildcard characters are fully supported.

Use

Use the TABLE command when you have multiple symbol tables loaded. In addition to the basic symbol table information, Visual SoftICE also displays whether exports are loaded for the table. Visual SoftICE supports symbol tables for any operating system image.

Visual SoftICE changes the current table to the table of the address context that the instruction pointer is in when the target stops. This is referred to elsewhere as on-demand symbol loading. Refer to ADDR and ADDSYM for more information about on-demand symbol loading.

Note: If you have set STICKYCONTEXT to *true*, then the automatic ADDR to the current instruction pointer's context will not occur when the master receives a stop event. To allow for automatic context switching, set STICKYCONTEXT to *false*. For more information, refer to the SET STICKYCONTEXT command.

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 table-name, that table becomes the current symbol table. If you specify the *-v* option, Visual SoftICE displays all tables regardless of whether symbols are currently loaded for them. If you specify the *-u* option, Visual SoftICE displays only the user-mode (Ring 3) tables. If you specify the *-s* option, Visual SoftICE displays only the system (Ring 0) tables.

You cannot use the TABLE command to load a symbol table if the matching image is not loaded on the target. In such an instance you must use the ADDSYM command to load the symbol table first.

Example

In the following example, the TABLE command, used without parameters, lists all loaded symbol tables. In the sample output, fib64_2.exe is highlighted because it is the current table.

```
SI>TABLE
Name           Version  Type   Gbl Exp Status
-------------------------------------------------
__USERNAMES__  00000001 User   Y   N   OK
fib64_2.exe    3B212A13 Symbol N   N   Matching PDB file
C:\user\fib64\debug_ia64\fib64_2.pdb
SICORE.SYS     3B1D06C5 Symbol Y   N   Can't find PDB symbol file.
```

In the following example, a table that is already loaded is specified as a parameter and Visual SoftICE makes it the current table.

```
SI>TABLE SICORE.SYS
```

In the following example, a table that is not yet loaded is specified as a parameter. Visual SoftICE loads it, and then makes it the current table.

```
SI>TABLE WDMAUD.SYS
```

See Also

ADDSYM, EXP, GETEXP, SET STICKYCONTEXT, SET SYMPATH, SET SYMTABLEAUTOLOAD, SYM

TCONFIG

TCONFIG used without any parameters displays all the information as though you used all of the parameters together.

Note: The target must be running for this command to succeed.

Syntax

```
TCONFIG
```

Use

Use the TCONFIG command to display all the information provided by all of the TCONFIG parameters as though you used them all at once.

Example

The following example displays all TCONFIG information:

```
SI>TCONFIG
Target Configurable Options:
Available Transports:
--------------------
    SI3C90X
    SISERIAL
--------------------
Current Transport:     SI3C90X
      KeepAlive Timeout: 0060 secs.   (Ranges: 0, 5 - 3600 Seconds)
--------------------
Core:
    StopOnBoot:        off
--------------------
SIAGENT SETTINGS
    Allow to copy files from a target                   on
   ...
    Allow to read\change debugger settings remotely     on
```

See Also

TCONFIG KEEPALIVE, TCONFIG STOPONBOOT, TCONFIG TRANSPORT

TCONFIG KEEPALIVE

Sets or displays a KEEPALIVE parameter of a transport driver on a target.

Note: The target must be running for this command to succeed.

Syntax

```
TCONFIG KEEPALIVE [value]
```

value Time in seconds. Range is 5 to 3600, and 0 disables the function.

Use

Use the TCONFIG KEEPALIVE command to configure the KEEPALIVE parameter of the transport driver on the target. You can disable KEEPALIVE by setting it to 0, or you can set its value to anything within the range of 5 to 3600 seconds. If you use TCONFIG KEEPALIVE without any parameters, Visual SoftICE returns the current value set.

Example

The following example disables KEEPALIVE for the transport driver on the target:

```
SI>TCONFIG KEEPALIVE 0
```

The following example sets KEEPALIVE for the transport driver on the target to 30 seconds:

```
SI>TCONFIG KEEPALIVE 30
```

See Also

TCONFIG, TCONFIG STOPONBOOT, TCONFIG TRANSPORT

TCONFIG STOPONBOOT

Sets or displays the STOPONBOOT parameter of the target debugger.

Note: The target must be running for this command to succeed.

Syntax

```
TCONFIG STOPONBOOT [on | off]
```

on	Enable the parameter.
off	Disable the parameter.

Use

Use the TCONFIG STOPONBOOT command to enable or disable the STOPONBOOT parameter of the target debugger. If you use TCONFIG STOPONBOOT without any parameters, Visual SoftICE returns the current value set.

Example

The following example enables the STOPONBOOT parameter:

```
SI>TCONFIG STOPONBOOT on
```

The following example disables the STOPONBOOT parameter of the transport driver on the target:

```
SI>TCONFIG STOPONBOOT off
```

See Also

TCONFIG, TCONFIG KEEPALIVE, TCONFIG TRANSPORT

TCONFIG TRANSPORT

Sets or displays a transport driver on a target.

Note: The target must be running for this command to succeed.

Syntax

```
TCONFIG TRANSPORT [transport-name]
```

transport-name The name of the transport driver you want to enable for the target.

Use

Use the TCONFIG TRANSPORT command to set or display the transport driver on a target. Using TCONFIG TRANSPORT without any parameters displays a list of available transport drivers on the target, and highlights the active one. Using it with a *transport-name* parameter sets that transport driver as the one used by the target.

Example

The following example displays a list of available transport drivers on the target:

```
SI>TCONFIG TRANSPORT
```

The following example sets SI8139 as the transport driver for the target:

```
SI>TCONFIG TRANSPORT SI8139
```

See Also

TCONFIG, TCONFIG KEEPALIVE, TCONFIG STOPONBOOT

THREAD

Displays information about a thread.

Syntax

Enumerate Process Threads

```
THREAD [PID | process-name | TID]
```

Enumerate All Threads

```
THREAD *
```

Display Thread Detail Data

```
THREAD -x [PID | process-name | TID]
```

Display a List of Objects the Thread is Waiting on

```
THREAD -w TID
```

Display a List of Threads with User-level Components

```
THREAD -u TID
```

Display Value of Thread's Registers

```
THREAD -r [-f frame] TID
```

-r	Display value of the thread's registers.
-f frame	Used only with *-r* to specify the frame stack number. For example -f 3.
-x	Display extended information for each thread.
-w	Display objects the thread is waiting on.
-u	Display threads with user-level components.
PID	Process-ID
TID	Thread handle or thread ID.
process-name	Process-handle or process-name.
*	Wildcard: displays every thread in the system.

Use

Use the THREAD command to obtain information about a thread.

- If you do not specify any options or parameters, it displays the summary information for all the threads in the active/current process in the debugger.
- If you specify a *process-type* as a parameter, it displays the summary information for all threads in that process.
- If you specify a *thread-type*, it displays detailed information for that thread.
- If you specify THREAD *, it displays the summary information for every thread in the system.

For the *-r* and *-fn* options, the registers shown are those that are saved on the thread context switches. If you use the *-r* option, the top stack frame's registers are displayed. The *-fn* option allows you to specify any other stack frame.

Output

For each thread, the following summary information is displayed:

TID	Thread ID.
Krnl TEB	Kernel Thread Environment Block.
StackBtm	Address of the bottom of the thread's stack.
StackTop	Address of the start of the thread's stack.
StackPtr	Thread's current stack pointer value.
User TEB	User thread environment block.
Process(Id)	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:

TID	Thread ID.
KTEB	Kernel Thread Environment Block.
Base Pri, Dyn. Pri	Threads base priority and current priority.

Mode Indicates whether the thread is executing in user or kernel mode.

Switches Number of context switches made by the thread.

Affinity Processor affinity mask of the thread. Bit positions that are set represent processors on which the thread is allowed to execute.

IP Address Address at which the thread will start executing when it is resumed or current IP if it is active.

The thread’s stack trace is displayed last.

Example

The following example displays extended information on the thread with ID 8:

```
SI>THREAD -x 8
----------------------------------------------------------------------
Tid             :  8
...
Name            :  System
Pid             :  4
```

See Also

IMAGE, STACK

TIMER

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 the output of TIMER when it is issued for a specific timer object:

```
SI>TIMER e000000086771ef0
----------------------------------------------------------------------
Address        :  e000000086771ef0
...
Name           :  AUTOENRL:MachineEnrollmentTimer
```

See Also

APC, DPC

TYPES

Lists all types in the current context, or lists all type information for the *type-name* parameter specified.

Syntax

```
TYPES [-base] [-enum] [-tdef] [-udt] [-f] [-v] [type-name]
```

-base Include base types in the search.

-enum Include enumerations in the search.

-tdef Include typedefs in the search.

-udt Include user-defined types in the search.

-f Member function mode. Display any member functions that exist for the type.

-v Verbose mode. Expand structures and classes to show members.

type-name List all type information for the type-name specified.

Use

If you do not specify a *type-name*, TYPES lists all the types in the current context. If you do specify a *type-name*, TYPES lists all the type information for the *type-name* parameter you specified. If you use the *-v* flag, TYPES expands the structures and classes to show members. You can filter the data returned by using combinations of the *-enum*, *-udt*, *-base*, and *-tdef* flags. Specifying none of the flags is equivalent to asking for all data.

Members are listed by offset within a parent structure, and parents are listed by name.

Example

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

```
SI>TYPES
Size Type Name Typedef
0x0004 ABORTPROC int stdcall (*proc) (void)
...
0x0004 BSTR unsigned short *
```

The following example displays all the types for the CTestHandles class, including its member functions.

```
SI>TYPES -v -f CTestHandles
Size     Offset   Tag        Name      Typedef
----------------------------------------------
000001d0 00000000 Composite  CTestHandles class CTestHandles
00000001 00000000 VTable     (null)     *
00000001 00000000 Function   CTestHandles CTestHandles void  (*)
...
00000004 0000000c Data       m_hAccel   struct HACCEL__*
00000004 00000044 Data       m_hMainMenu struct HMENU__*
00000004 00000048 Data       m_hWnd     struct HWND__*
...
Types query returned 92 matches.
```

The following example displays only the typedefs that begin with B in the current table.

```
SI>TYPES -tdef B*
Size     Offset   Tag        Name      Typedef
----------------------------------------------
00000001 00000000 Typedef    BCHAR      BCHAR unsigned char
00000004 00000000 Typedef    BITVEC     BITVEC unsigned int
00000004 00000000 Typedef    BOOL       BOOL int
00000001 00000000 Typedef    BOOLEAN    BOOLEAN unsigned char
00000001 00000000 Typedef    BYTE       BYTE unsigned char
00000001 00000000 Typedef    bool       bool unsigned char
Types query returned 6 matches.
```

See Also

LOCALS, WL

UNASSEMBLE

Unassemble instructions.

Syntax

```
UNASSEMBLE [+i] [-a] [-b] address [L length]
```

or

```
U [+i] [-a] [-b] address [L length]
```

+i	Tell the disassembler to display instructions.
-a	Tell the disassembler to hide addresses.
-b	Tell the disassembler to display the output as bundles.
address	Segment offset or selector offset.
L length	Number of instruction bytes (x86) or bundles (IA64).

Use

The UNASSEMBLE command displays unassembled code at the specified address.

If you do not specify the address, the UNASSEMBLE command unassembles the code starting at the address where you last unassembled code. If you have never executed the UNASSEMBLE command, you *must* specify an address.

If you specify a length, Visual SoftICE unassembles the specified number of instructions bytes, rounding up to the next full instruction. Generally, this works out to be one instruction line per instruction byte of length specified. If you specify a length on IA64, Visual SoftICE unassembles the specified number of instructions bundles, with one bundle containing a minimum of two, and a maximum of three, instruction lines.

If the disassembly contains the current Instruction Pointer (IP), the UNASSEMBLE command highlights it. If the current IP instruction references a memory location, the UNASSEMBLE command displays the contents of the memory location at the end of the code line.

If any of the referenced memory addresses in the disassembly have symbolic names, the symbol displays instead of or in addition to the hexadecimal address. If an instruction is located at a code symbol, the symbol name displays.

Example

The following example unassembles instructions beginning at 10h bytes before the current address:

```
SI>UNASSEMBLE . - 10
```

UNLOAD

Unload symbols for an image file.

Syntax

```
UNLOAD [image-name]
```

image-name Name of the image for which you want to unload symbols.

Use

Use the UNLOAD command to unload symbols for an image.

Example

The following example unloads symbols for the `myprogram.exe` image:

```
SI>UNLOAD myprogram.exe
```

See Also

ADDSYM, DELSYM, FILE, GETEXP, LOAD, RELOAD, SET SYMSRVSEARCH, SET SYMTABLEAUTOLOAD

VERSION

Display the Visual SoftICE version number for both the master and any connected target machine.

Syntax

```
VERSION [-x]
```

-x Display extended version information.

Use

Use the VERSION command to display version information for the Master and Target. Use the -*x* flag to display extended version information for Visual SoftICE and the OS data files, allowing Compuware Technical Support to better diagnose any problems you may encounter.

Example

The following example displays the Visual SoftICE extended version information for the master and any connected target machine:

```
SI>VERSION -x
[Master]: SIDE.EXE (Visual SoftICE GUI) Ver. 1.0.0 (build 470)
[Target]: SICORE.SYS Ver. 1.0.455 (IA32(x86) Windows NT/XP v5.1)
[Additional Information]:
OSI Information for (osinfo.dat)
osi.copyrightmessage       = Copyright 2002 - Numega Labs
osi.versionstr             = 1
osi.revisionstr            = Supported MS Os
osi.date_built             = 05/17/02
osi.time_built             = 13:43:24
osi.builtby                = NuMega Labs
osi.description            = Supported MS OSs
osi.ulNumOfStructsDefined  = 398
osi.ulNumOfElementsDefined = 5420
osi.ulNumOfPatchesDefined  = 101
OSI Information for (osinfob.dat)
osi.copyrightmessage       = Copyright 2002 - Numega Labs
osi.versionstr             = 1
osi.revisionstr            = MS Beta Os
osi.date_built             = 05/17/02
osi.time_built             = 13:43:27
osi.builtby                = NuMega Labs
osi.description            = MS Beta OSs
osi.ulNumOfStructsDefined  = 471
osi.ulNumOfElementsDefined = 7192
osi.ulNumOfPatchesDefined  = 22
```

WATCH

Add a watch expression.

Syntax

`WATCH` *`expression`*

Use

Use the WATCH command to display the results of expressions. Visual SoftICE determines the size of the result based on the expression's type information. If Visual SoftICE cannot determine the size, DWORD is assumed. Every time Visual SoftICE stops, 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.

If the expression being watched goes out of scope, Visual SoftICE displays the following message: "Error evaluating expression".

There can only be one Watch page open.

Example

The following example creates an entry in the Watch page for the variable `hInstance`.

```
SI>WATCH hInstance
```

WB

Open or switch to the Breakpoint page.

Syntax

```
WB [name]
```

name	Specify a name to give the Breakpoint page.

Use

WB opens a new Breakpoint page, or switches to the page if one is already opened. If you specify a name, the created page will take that name only if the name is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following command opens the Breakpoint page, if it is closed, and names it BREAKPOINTS:

```
SI>WB BREAKPOINTS
```

See Also

WC, WD, WE, WF, WG, WI, WL, WP, WR, WS, WT, WX

WC

Open a Source or Disassembly page on a specified address.

Syntax

```
WC [address]
```

address	Any address.

Use

WC opens the source file containing the specified address in a Source page. If the source file for the specified address does not exist, Visual SoftICE opens a Disassembly page on the address.

If you specify an address parameter, such as EIP or some function name, Visual SoftICE places focus at the address in the Source or Disassembly page. WC first checks if any source files contain the address. If yes, Visual SoftICE brings up that source file (if the file is not opened yet) and points to the line of code that the address points to. If no source file can be found, Visual SoftICE brings up the Disassembly page and points to the address (regardless of the autofocus setting).

Example

The following example opens a Source or Disassembly page at the address location of the EIP:

```
SI>WC EIP
```

See Also

WB, WD, WE, WF, WG, WI, WL, WP, WR, WS, WT, WX

WCONNECT

Wait for a connection to a target machine.

Syntax

```
WCONNECT com# [-baud #] [-rt #] [-r #]
WCONNECT nnn.nnn.nnn.nnn [password] [-rt #] [-r #]
WCONNECT hostname [password] [-rt #] [-r #]
```

com#	Specifies the COM port for serial connections. COM1 through COM4 are valid. When connecting through a serial connection, you can also specify the baud rate, retry timeout, and retry count.
IP	The IP address of the target (`nnn.nnn.nnn.nnn`). When connecting through an IP address, you can also specify a password, retry timeout, and retry count.
hostname	The host name of the target. DNS matches the host name to its IP address and connects through the IP address. You can specify a partial host name and Visual SoftICE will match it to the complete host name if it can. You can also specify a password, retry timeout, and retry count.
-baud #	Specifies the baud rate for serial connections. Default is 115200.
password	Specifies a password to use to connect via IP address or hostname if the target is password-protected.
-rt #	Specifies the retry timeout value. Default is 20ms.
-r #	Specifies the retry count value. Default is 5.

Note: Numeric values (baud, retry timeout, retries) are entered in decimal.

Use

Use the WCONNECT command to wait for a connection to the target machine. You can connect to the target through serial connection or by IP address. When connecting by IP address, you can supply either the IP address, or enough of a recognized host name for Visual SoftICE to complete a DNS lookup.

Example

The following example waits for a connection to the target using its IP address:

```
SI>WCONNECT 255.255.255.0
Connected to:
  Name             : mytarget-IA64
Processor      : IA64-Itanium
Stepping       : 0
Processor Count: 2
Operating Sys. : Windows XP-64 Ver. 5.1 Build 2600
Target Agent   : Connected (Active)
```

The following example waits for a connection to the target using its hostname:

```
SI>WCONNECT testxp
Connected to:
Name           : TESTXP
Processor      : IA32(x86)-Pentium III
Stepping       : 1
Processor Count: 1
Operating Sys. : Windows NT/XP Ver. 5.1 Build 2600
Target Service : Not available.
SI>reboot
Target has entered sleep mode (5) [Shutdown Reset]
Target shutting down. Disconnecting...
SI>WCONNECT testxp
Attempting to resolve the specified name to an IP address...
...........................................
Connected to:
Name           : TESTXP
Processor      : IA32(x86)-Pentium III
Stepping       : 1
Processor Count: 1
Operating Sys. : Windows NT/XP Ver. 5.1 Build 2600
Target Service   : Not available.
DriverStudio Service has started on target.
```

See Also

CLOSE, CONNECT, DISCONNECT, NETFIND, OPEN

WD

Open a Memory page.

Syntax

```
WD [address] [name]
```

address	The start address for displaying memory.
name	Specify a name to give the Memory page.

Use

WD opens the Memory page. If a Memory page is already open, WD opens another one. If you specify an address, Visual SoftICE opens the Memory page and displays memory starting at that address. If you specify a name, the created page will take that name only if the name is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following example opens a Memory page and names it `MEMORY`:

```
SI>WD MEMORY
```

See Also

WB, WC, WE, WF, WG, WI, WL, WP, WR, WS, WT, WX

WE

Open an Event page.

Syntax

```
WE [name]
```

name Specify a name to give the Event page.

Use

WE opens a new Event page, or switches to the page if one is already opened. If you specify a name, the created page will take that name only if the name is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following command opens the Event page, if it is closed, and names it EVENTS:

```
SI>WE EVENTS
```

See Also

WB, WC, WD, WF, WG, WI, WL, WP, WR, WS, WT, WX

WF

Display the floating point registers in either floating point or MMX format.

Syntax

```
WF
```

Use

WF opens the Floating Point Registers page.

The default display format is double-precision floating point representation. To change the display format to any of the supported raw and floating point options, right-click on the page and select Data Format Preferences from the pop-up menu.

Example

The following example shows the use of the WF command to open the Floating Point Registers page:

```
SI>WF
```

See Also

WB, WC, WD, WE, WG, WI, WL, WP, WR, WS, WT, WX

WG

Open a Debug Message page.

Syntax

```
WG [name]
```

name	Specify a name to give the Debug Message page.

Use

WG opens a new Debug Message page. If you specify a name, the created page will take that name only if it is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following command opens a Debug Message page and names it DEBUG:

```
SI>WG DEBUG
```

See Also

WB, WC, WD, WE, WF, WI, WL, WP, WR, WS, WT, WX

WHAT

Determine what kind of object is at a memory address.

Syntax

```
WHAT [address]
```

address	Any target address, name, or 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. It will search for a symbol, driver, device, and all other possibilities, and display all the **possible** matches. Where appropriate, type identification of a match is expanded to indicate relevant information, such as a related process or thread.

When passing a name to the WHAT command, the name 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.

When passing an expression to the WHAT command, the expression is typically something that cannot be considered a name. That is, it is a number, a complex expression (an expression containing operators, such as `explorer+0`), or a register. Although a register looks like a name, registers are special cases of expressions since this usage is much more common. For example, Visual SoftICE interprets `eax` in `WHAT eax` as an expression-type. Symbol names are treated as names, and are correctly identified by the WHAT command as symbols.

Because the rules for determining name and expression types can be ambiguous at times, you can force a parameter to be evaluated as a name by placing it in quotes. To force Visual SoftICE to interpret a parameter that may be viewed as a name as a symbol, use SYMBOL(XXX). To force Visual SoftICE to interpret a parameter that may be viewed as a name as a register name, use REG(XXX).

Examples

The following is an example of using the WHAT command on the explorer.exe process, and the resulting output.

```
SI>WHAT explorer.exe
Process: Explorer(0x5D8), Base Address: 0x1000000, Path: C:\WINDOWS\Explorer.EXE
Image: Explorer.EXE, Base Address: 0x1000000, Process Id: 0x5D8
```

The following is an example of using the WHAT command on the address e0000000869919c0.

```
SI>WHAT e0000000869919c0
E0000000869919C0 (-2305843006955513408) Found 1 Matches:
KEvent:
Address: e0000000869919c0, Type: 0, SignalState: 1, WaitListHead: e0000000869919c8, Name: ShellReadyEvent,
```

The following is an example of using the WHAT command on the address e0000000831c8820.

```
SI>WHAT e0000000831c8820
E0000000831C8820 (-2305843007014008800) Found 1 Matches:
Symbol:
e0000000831c8820 = ntoskrnl!IopRootDeviceNode (Public)
```

The following is an example of using the WHAT command on the address e00000008306d7a0.

```
SI>WHAT e00000008306d7a0
E00000008306D7A0 (-2305843007015430240) Found 2 Matches:
Symbol:
e00000008306d7a0 = ntoskrnl!.KeInsertQueueDpc (Public)
KEvent:
Address: e00000008306d7a0, Type: 0, SignalState: 80, WaitListHead: 8cfc670180006200, Name: ,
```

WI

Open a Command (input) page.

Syntax

```
WI [name]
```

name	Specify a name to give the Command page.

Use

WI opens a new Command page. If you specify a name, the created page will take that name only if it is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following command opens a Command page and names it `CMD`:

```
SI>WI CMD
```

See Also

WB, WC, WD, WE, WF, WG, WL, WP, WR, WS, WT, WX

WINERROR

Display header-defined mnemonics for Win32/64 error codes.

Syntax

`WINERROR` *`code`*

code The Win32/64 error code you want a mnemonic returned for.

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 Win32/64 error code.

Example

The following example shows the WINERROR command returning the mnemonic for the error code `0x103`:

```
SI>WINERROR 0x103
ERROR_NO_MORE_ITEMS
```

WL

Open a Local Variables page.

Syntax

```
WL [TID] [stack-frame]
```

TID Thread ID for the current process.

stack-frame Stack frame index for the current thread.

Use

WL opens the Local Variables page. If a Local Variables page is already open, WL activates that page. You can only have one Local Variables page open.

Note: From within the Locals page, you can expand structures, arrays, and character strings to display their contents. Simply double-click the item you want to expand. Note that expandable items are indicated with a plus sign (+).

Example

The following example opens a Local Variables page with a TID for the current process:

```
SI>WL 128
```

See Also

LOCALS, TYPES, WB, WC, WD, WE, WF, WG, WI, WP, WR, WS, WT, WX

WMSG

Display the names and message numbers of Windows messages.

Syntax

```
WMSG [partial-name | msg-number]
```

partial-name Windows message name or the first few characters of a Windows message name. If multiple Windows messages match the partial-name, then all messages that start with the specified characters display.

msg-number Hexadecimal message number of the message. Only the message that matches the message number displays.

Use

WMSG 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.

```
SI>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.

```
SI>WMSG 111
0111 WM_Command
```

WP

Opens the Process List page.

Syntax

```
WP [name]
```

name	Specify a name to give the Process List page.

Use

You can use WP to open the Process List page. If the Process List page is already open, then the WP command activates it. You can only have one Process List page open.

Example

The following command opens the Process List page and names it PROCESSES:

```
SI>WP PROCESSES
```

See Also

WB, WC, WD, WE, WF, WG, WI, WL, WR, WS, WT, WX

WR

Open a Register page.

Syntax

```
WR [name]
```

name	Specify a name to give the Register page.

Use

WR opens the Register page. If a Register page is already open, WR opens another one. If you specify a name, the created page will take that name only if it is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

The Register window displays the register set and the processor flags.

Example

The following command opens the Register page and names it REGISTER:

```
SI>WR REGISTER
```

See Also

WB, WC, WD, WE, WF, WG, WI, WL, WP, WS, WT, WX

WS

Opens the Stack page.

Syntax

```
WS [name]
```

name	Specify a name to give the Stack page.

Use

WS opens the Stack page. If you specify a name, the created page will take that name only if it is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following command opens the Stack page and names it STACK:

```
SI>WS STACK
```

See Also

WB, WC, WD, WE, WF, WG, WI, WL, WP, WR, WT, WX

WT

Open a Text scratch page.

Syntax

```
WT [name]
```

name	Specify a name to give the Text scratch page.

Use

WT opens a new Text scratch page. If you specify a name, the created page will take that name only if it is unique. If another page already has that name, Visual SoftICE appends a sequence number to the name.

Example

The following command opens a Text scratch page and names it SCRATCH:

```
SI>WT SCRATCH
```

See Also

WB, WC, WD, WE, WF, WG, WI, WL, WP, WR, WS, WX

WX

Open the XMM register page.

Syntax

```
WX
```

Use

On supported platforms, you can use the WX command to open a page containing the value of the XMM registers `XMM0` through `XMM7`.

Example

The following example displays the XMM register page:

```
SI>WX
```

See Also

WB, WC, WD, WE, WF, WG, WI, WL, WP, WR, WS, WT

XFRAME

Display exception handler frames that are currently installed.

Note: Available on x86 only.

Syntax

```
XFRAME [thread-type]
```

thread-type	Value that Visual SoftICE recognizes as a thread.

Use

Exception frames are created by Microsoft's Structured Exception Handling API (SEH). Handlers are instantiated on the stack, so they are context-specific.

When an exception handler is installed, information about it is recorded in the current stack frame. This information is referred to as an ExceptionRegistration. The XFRAME command understands this information, and walks backwards through stack frames until it reaches the top-most exception handler. From there it begins displaying each registration record up to the currently active scope. From each registration, it determines if the handler is active or inactive; its associated "global exception handler;" and, if the handler is active, the SEH type: try/except or try/finally. In the case of active exception handlers, it also displays the exception filter or finally handler address.

Note: The global exception handler is actually an exception dispatcher that uses information within an exception scope table to determine which exception handler, if any, 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 the optional parameter, Visual SoftICE reverts to the last active context 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, Visual 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. Because 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 paged in for the specified thread, the XFRAME will not be able to complete the stack walk.

Output

For each exception frame that is installed, XFRAME displays the following information:

xFrame	Address of the ExceptionRegistration. This value is stack-based.
xHandler	Address of the global exception handler which dispatches the exception to the appropriate try/ except/finally filter/handler.
xTable	Address of the scope table used by the global exception handler to dispatch exceptions.
xScope	Index into the xTable for the currently active exception handler. If this value is -1, the exception handler is installed, but is inactive and will not trap an exception.

Example

The following example illustrates the use of the XFRAME command to display information about the exception handler frames for the currently active thread.

```
SI>XFRAME
xFrame    ScopeTable  Scope     Type    xHandler  Status
--------------------------------------------------------
0012ffe0  77e95a18    0         MS SEH  77e9bb86
Enclosing Level  Type        Filter    Handler
-------------------------------------------------
0                try/except  77ea5168  77ea5179
xFrame    ScopeTable  Scope     Type    xHandler  Status
--------------------------------------------------------
0012ffb0  00425498    0         MS SEH  00402238
Enclosing Level  Type        Filter    Handler
-------------------------------------------------
0                try/except  004027a8  004027c3
xFrame    ScopeTable  Scope     Type    xHandler  Status
--------------------------------------------------------
0012fecc  0012ff80    2         C++ EH  00413d60
0012fe64  00425250    2         MS SEH  00402238
Enclosing Level  Type         Filter    Handler
--------------------------------------------------
0                try/except   0155c318  01560f74
1               try/except   5f8ad3ec  00000000
2              try/finally  00000000  004016c5
```

ZAP

Replace an embedded interrupt 1 or 3 with a NOP.

Syntax

ZAP

Use

The ZAP command replaces an embedded interrupt 1 or 3 with the appropriate number of NOP instructions. This is useful when the INT 1 or INT 3 is placed in code that is repeatedly executed, and you no longer want Visual SoftICE to stop. 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 the embedded interrupt 1 or interrupt 3 with a NOP instruction.

```
SI>ZAP
```

Index