e Language Quick Reference

November 2003

This card contains selected ***e*** constructs. For complete ***e*** syntax, see the ***e** Language Reference*.

Abbreviations:

arg - argument	inst - instance
bool - boolean	num - number
enum - enumerated	TCM - time-consuming method
expr - expression	TE - temporal expression

Predefined Types

bit // unsigned integer with value 0 or 1 (default: 0)

byte // unsigned integer in the range 0-255 (default: 0)

int // 32-bit signed integer (default: 0)

uint // 32-bit unsigned integer (default: 0)

int | uint (bits: *n* **| bytes:** *n* **)** // n-bit or n-byte signed int or uint

bool // one-bit boolean (0 = FALSE, 1 = TRUE) (default: FALSE)

list [**(key:** *field-name***)**] **of** *type*
// a list of elements of the specified type (default: empty)

string // strings are enclosed in quotes: “my string” (default: NULL)

Type Conversion

expr **=** *expr***.as_a(** *type* **)**

User-Defined Types   Statements

struct *struct-type* [**like** *base-struct-type*] **{** struct members **};**

unit *unit-type* [**like** *base-unit-type*] **{** unit members **};**

type *type-name* **:** [**u**]**int (bits:** *n* **| bytes:** *n* **);** // defines a scalar type

type *type-name* **: [** *name* [**=***n*], ... **];** // defines an enumerated type

extend *type-name* **: [** *name* [**=***n*], ... **];** // extends an enumerated type

extend *struct-type|unit-type* **{** additional struct or unit members **};**
// extends a struct or unit

Struct and Unit Members

fields	constraints	when conditions
methods and TCMs	cover groups	events
temporal struct|unit members	preprocessor directives	

Fields   Struct and Unit Members

[**!**][**%**]*field-name* **:** *type*; // ! = do not generate, % = physical field

field-name[*n*] **: list of** *type*; // creates a list with n elements

field-name **:** *unit-type* **is instance;** // for units only, not structs

Conditional Extensions using When   Struct and Unit Members

type *enum-type*: **[***name1*, *name2*, ...**];**
struct|unit *struct-type|unit-type* **{**
field-name **:** *enum-type*;
when *name1 struct-type|unit-type* **{** additional members **};**
};
extend *name1 struct-type|unit-type* **{** ... **};**

Ports   Struct and Unit Members

port-name: *dir* [**buffer_port** | **simple_port**] **of** *data-type* **is instance;**

port-name: *dir* **event_port is instance;**

keep [**soft**] *port_inst.attribute*() == *value;*

keep bind(*inst-name1*, *inst-name2* **|external |empty | undefined);**

Encapsulation   Statements, Struct and Unit Members

package *package-name*; [**package**] **type** *type*;

package | protected | private *struct-member-definition*;

Constraints   Struct and Unit Members

keep [**soft**] *bool-expr*; // for example, keep field1 <= MY_MAX

keep [**soft**] *field-name* **in [***range***];** // example: keep field1 in [0..256]

keep *bool-expr1* **=>** *bool-expr2*; // bool-expr1 implies bool-expr2

keep [**soft**] *field-name* **in** *list*;

keep *list***.is_all_iterations(** *field-name* **);**

keep *list1***.is_a_permutation(** *list2* **);**

keep *list*[*index*].*field-name constraint-expr*;

keep for each (*item* **) in** *list* **{** [**soft**] *bool-expr*; ... **};**

keep all of {*constraint-expr*; ... **};**

keep soft *bool-expr* **== select {** *weight* **:** *value*; ... **};**

keep [**soft**] **gen (** *item-a* **) before (** *item-b* **);**

keep *gen-item***.reset_soft();** // ignore soft constraints on gen-item

keep *field-name***.hdl_path() ==** “*string*” **;** //field-name is unit instance

keep soft *bool-expr* **== select {** *weight* **:** *value*; ... **};**

keep gen_before_subtypes(*determinant-field*: *field*, ...**);**

Predefined Methods of All Structs   Struct and Unit Members

run()	**extract()**	**check()**	**finalize()**
init()	**pre_generate()**	**post_generate()**	
copy()	**do_print()**	**print_line()**	**quit()**

Methods and TCMs   Struct and Unit Members

*regular-method***(** [*arg* **:** *type*, ...] **)** [**:** *return-type*] **is {** *action*; ... **};**

*TCM***(**[*arg* **:** *type*, ...] **)** [**:** *return-type*] **@***event-name* **is {** *action*; ... **};**

*method***(***arg* **:** *type*, ... **)** [**:** *return-type*] [**@***event-name*] **is also|first|only {** *action*; ... **};**

Variable Declarations and Assignments   Actions

var *var-name* **:** *type*; **var** *var-name* **: =** *value*;

var-name = *expr* ; // e.g. field-name=expr, var-name=method()

Conditional Procedures   Actions

if *bool-expr* [**then**] **{** *action*; ... **}**
[**else if** *bool-expr* [**then**] **{** *action*; ... **}**] [**else {** *action*; ... **}**] ;

case { *bool-expr*[**:**] **{** *action*; ... **}** ; [**default**[**:**] **{** *action*; ... **}** ;] **};**

case *expr* **{** *value*[**:**] **{** *action*; ... **}** ; [**default**[**:**] **{** *action*; ... **}** ;] **};**

Checks   Actions

check that *bool-expr* [**else dut_error(** ... **)**];

Loops   Actions

for *i* **from** *expr* [**down**] **to** *expr* [**step** *expr*] [**do**] **{** *action*; ... **};**
for each [*struct-type*] (*list-item*) [**using index** (*index-name*)]
in [**reverse**] *list* [**do**] **{** *action*; ... **};**
for each [**line**] [(*line-name*)] **in file** *file-name* [**do**] **{***action*; ... **};**
while *bool-expr* [**do**] **{** *action*; ... **};**

break; **continue;**

Invoking Methods and TCMs   Actions

*TCM2***()@***event-name* **is {** *TCM1***();** *method***();};** // calling methods

*method1***() is {** *method2***();** *method3***(); };** // calling methods

*method***() is { start** *TCM***();};** // starting a TCM on a separate thread

Note: A TCM can only be *called* from another TCM. However, a TCM can be *started* from a regular method or from another TCM.

Operators

Operator precedence is left to right, top to bottom in the list

[ ]	list indexing	**[..]**	list slicing
[:]	bit slicing	*f***()**	method or routine call
.	field selection	**in**	range list
{... ; ...}	list concatenation	**%{... , ...}**	bit concatenation
~	bitwise not	**!, not**	boolean not
+, -	unary positive, negative	***, /, %**	multiply, divide, modulus
+, -	plus, minus	**>>, <<**	shift right, shift left
<, <=, >, >=	boolean comparison	**is** [**not**] **a**	subtype identification
==, !=	boolean equal, not equal	**===,!==**	Verilog 4-state compare
~, !~	string matching	**&, |, ^**	bitwise and, or, xor
&&, and	boolean and	**||, or**	boolean or
!, not	boolean not	**=>**	boolean implication
a **?** *b* **:** *c*	conditional “if a then b, else c”		

Simulator Interface   Statements and Unit Members

verilog function ’*HDL-path*’(*params*) **:** *n*; // n is result size in bits

verilog import *file-name*; // statement only

verilog task ’*HDL-path*’(*params*);

verilog time *Verilog-timescale*; // statement only

vhdl driver ’*HDL-path*’ **using** *option*, …; // unit member only

vhdl function ’*designator*’ **using** *option*, …;

vhdl procedure ’*identifier*’ **using** *option*, …;

vhdl time *VHDL-timescale*; // statement only

Generation On the Fly   Actions

gen *gen-item* [**keeping {** [**soft**] *constraint-bool-expr* ; ... **}**];

Events

event *event-name* [**is** [**only**] *TE*]; // struct or unit member

emit [*struct-inst*.]*event-name*; // action

Predefined Events

sys.any	*struct-inst*.**quit**

Temporal Struct and Unit Members

on *event-name* **{** *action*; ... **}** ;

expect|assume [*rule-name* **is** [**only**]] *TE*
[**else dut_error(** "*string*", *expr*, ... **)**];

Temporal Expressions (TEs)

Basic Temporal Expressions

@[*struct-inst*.]*event-name*	// event instance
change|fall|rise('*HDL-path*'**) @sim**	// simulator callback annotation
change|fall|rise(*expr***)**	**true(***bool-expr***)** **cycle**

Boolean Temporal Expressions

TE1 **and** *TE2*	*TE1* **or** *TE2*	**not** *TE*	**fail** *TE*

Complex Temporal Expressions

TE **@**[*struct-inst*.]*event-name*	// explicit sampling
{ *TE*; *TE*; ... **}**	// sequence
TE1 **=>** *TE2*	// if TE1, then TE2 follows
TE **exec {** *action*; ... **}**	// execute when TE succeeds
[*n* **]** [***** *TE*]	// fixed repeat
{ ... ; **[** [*n*]**..**[*m*] **]** [***** *TE*]; *TE*; ... **}**	// first match repeat
~[[*n*]**..**[*m*] **]** [***** *TE*]	// true match repeat
delay(*expr***)**	**detach(***TE***)**
consume(@[*struct-inst*.]*event-name* **)**	

Time-Consuming Actions

wait [[**until**] *TE*];	**sync** [*TE*];

Lock and Release, Semaphores — Predefined Structs and Methods

```
struct struct-type {
    locker-expr: locker;
    TCM() @event-name is {
        locker-expr.lock();
        ...
        locker-expr.release(); };
};
```

```
struct struct-type {
    sem-expr: semaphore;
    TCM() @event-name is {
        sem-expr.up();
        ...
        sem-expr.down(); };
};
```

Packing and Unpacking Pseudo-Methods

expr **= pack(** *pack-options*, *expr*, … **)**

unpack(*pack-options*, *value-expr*, *target-expr* [, *target-expr*, ...] **)**

Printing — Action

print *expr*[,…] [**using** *print-options*] ;

Predefined Routines — Actions

Deep Copy and Compare Routines

deep_copy(*expr* : struct-type**)** : struct-type

deep_compare[_physical](*inst1*: struct-type, *inst2*: struct-type,
max-diffs: int**)**: list of string

Output Routines

out ("*string*", *expr*, ...**);**	**out (** *struct-inst* **);**

outf ("*string* **%c** ...", *expr* **);** // c is a conversion code: s, d, x, b, o, u

Selected Configuration Routines

Note: Categories for these routines are listed in "Configuration Commands" in the Specman Elite Quick Reference.

set_config(*category*, *option*, *option-value* **)**

get_config(*category*, *option* **);**

Selected Arithmetic Routines

min|max (*x*: int, *y*: int**)**: int	**abs(***x*: int**)**: int
ipow(*x*: int, *y*: int**)**: int	**isqrt(***x*: int**)**: int
odd|even (*x*: int**)**: bool	**div_round_up(***x*: int, *y*: int**)**: int

Bitwise Routines

expr.**bitwise_and\|or\|xor\|nand\|nor\|xnor(***expr*: int\|uint**)**: bit

Selected String Routines

appendf(*format*, *expr*, ...**)**: string	**append(***expr*, ...**)**: string
expr. **to_string()**: string	**bin|dec|hex(***expr*, ...**)**: string

str_join(*list*: list of string, *separator*: string**)**: string

str_match(*str*: string, *regular-expr*: string**)**: bool

str_replace(*str*:string, *regular-expr*:string, *replacement*:string**)**:string

str_split(*str*: string, *regular-expr*: string**)**: list of string

Selected Operating System Interface Routines

system("*command***")**: int	**date_time()**: string

output_from("*command***")**: list of string

output_from_check("*command***")**: list of string

get_symbol(*UNIX-environment-variable*: string**)** : string

files.write_string_list(*file-name*: string, *list*: list of string**)**

Stopping a Test

stop_run(); // stops the simulator and invokes test finalization

On-the-Fly Memory Management

do_otf_gc()

Preprocessor Directives — Statements, Struct Members or Actions

#define [']*macro-name* [*replacement*]

#if[n]def [']*macro-name* **then {***string***}** [**#else {***string***}**] ;

List Pseudo-Methods

Selected List Actions

add[0](*list-item* : list-type**)**	**add[0](***list* : list**)**
clear()	**delete(***index* : int**)**
pop[0]() : list-type	**push[0](***list-item* : list-type**)**
insert(*index* : int, *list* : list | *list-item* : list-type**)**	

Selected List Expressions

size() : int	**top[0]()** : list-type
reverse() : list	**sort(***expr* : expr**)** : list
sum(*expr* : int**)** : int	**count (***expr* : bool**)** : int
exists(*index* : int**)** : bool	**has(***expr* : bool**)** : bool
is_empty() : bool	**is_a_permutation(***list*: list**)** : bool
all(*expr* : bool**)** : list	**all_indices(***expr* : bool**)** : list of int
first(*expr* : bool**)** : list-type	**last(***expr* : bool**)** : list-type
first_index(*expr* : bool**)** : int	**last_index(***expr* : bool**)** : int
key(*key-expr* : expr**)** : list-item	**key_index(***key-expr* : expr**)** : int
max(*expr* : int**)** : list-type	**max_value(***expr* : int**)** : int | uint
min(*expr* : int**)** : list-type	**min_value(***expr* : int**)** : int | uint
swap(*small* : int, *large* : int**)** : list of bit	
crc_8|32(*from-byte* : int, *num-bytes* : int**)** : int	
unique(*expr* : expr**)** : list	

Coverage Groups and Items — Struct and Unit Members

cover *cover-group* [**using** [**also**] *cover-group-options*] **is** [**empty**] [**also**] **{**
item *item-name* [**:** *type* **=** *expr*] [**using** [**also**] *cover-item-options*];
cross *item-name1*, *item-name2*, ... ; **transition** *item-name*;
};

To enable coverage, extend the **global** struct as follows:
setup_test() is also {set_config(cover, mode, *cover-mode***)}**

Coverage Group Options

text = *string*	**weight =** *uint*	**no_collect**	**radix = DEC|HEX|BIN**
count_only	**global**	**when =** *bool-expr*	
external=surecov		**agent_options=***SureCov options*	

Coverage Item Options

text = *string*	**when =** *bool-expr*	**weight =** *uint*
no_collect	**radix=DEC|HEX|BIN**	**name** *name*
at_least = *num*	**ignore | illegal =** *cover-item-bool-expr*	
no_trace	**ranges=range(** **[** *n*..*m* **]**, *sub-bucket-name*, *sub-bucket-size*, *at-least-number* **);**	
per_instance	**agent_options=***SureCov options*	

Specman Elite
Quick Reference

November 2003

This card contains selected Specman Elite commands and procedures. For more information, see the *Specman Elite Command Reference*.

Abbreviations:	dir - directory	expr - expression
	inst - instance	num - number

General Help

help *command* [**syntax**]	**apropos** *command* [**syntax**]
Specview **Help** button	Specview **Vadvisor** button

Creating an HDL Stub File

write stubs -verilog | -qvh | -ncvhdl | -spd [*file-name*]

```
specman -command “load top.e; write stubs -verilog”
    // creates stub file named specman.v for most Verilog simulators
specman -command “load top.e; write stubs -qvh my_stub.vhd”
    // creates stub file for ModelSim VHDL named my_stub.vhd
```

Compiler Script

```
%sn_compile.sh
    // use with no arguments to display compiler script options
%sn_compile.sh top.e
    // create an executable named “top” with compiled top.e module
```

NC Simulator or ModelSim

```
%sn_compile.sh top.e -sim ncvlog
    // creates a Specman Elite executable named “ncvlog_top” that
    // includes the compiled top.e module and NC Simulator (Verilog)
%sn_compile.sh top.e -shlib
    // creates a library that includes top.e and ModelSim
```

VCS

```
%sn_compile.sh -sim vcs -vcs_flags “file1.v ... specman.v” top.e
    // creates a Specman Elite executable named “vcs_top” that
    // includes VCS, compiled top.e and Verilog source files
```

Incremental Compilation Command Sequence

1. sn_compile.sh -e my_dir -t . first.e
2. sn_compile.sh -s my_dir/first -t . next.e
3. sn_compile.sh -s my_dir/next -t . last.e

Switching between Specman Elite and Simulator Prompts

```
<Cntl>-<Return>  // switch from simulator prompt to Specman Elite
                 // in text mode (no simulator GUI is being used)
$sn ;     // switch from Verilog-XL or VCS to Specman Elite
sn        // switch from ModelSim to Specman Elite
call sn  // switch from NC Simulator to Specman Elite
<Return>    // switch from Specman Elite back to the simulator
```

Specman Elite Commands from Simulator Prompt

Verilog-XL or **VCS**: $sn(“command”); **ModelSim**: sn “command”

NC Simulator: call sn {“command”}

Simulator-Related Commands

show functions	// Verilog and VHDL
show tasks [**and functions**]	// Verilog
show procedures	// VHDL
show subprograms	// VHDL
show defines [**-v**] [**-e**] [" [`]*macro-name*"]	// Verilog defines

Starting Specman Elite or the Specview GUI

Starting Specman Elite in Text Mode

specman [-p[re_commands] *commands* …]
[-c[ommands] *commands* …]

```
Example:
specman -p "config print -radix = HEX" -p "load top"
    // Starts Specman Elite, sets print radix to hex, and loads top.e
```

Starting the Specview GUI

specview [-p[re_commands] *commands* …]
[-c[ommands] *commands* …] [*integrated-executable parameters*]

```
Example:
specview xl_specman +gui -s xor.v specman.v
    // Starts Specview along with the Verilog-XL GUI, loads the xor.v
    // file and the specman.v stubs file
```

Running from Compiled Executables

%specsim [-pre-commands *command* ...] [-commands *command* ...]
[*integrated-executable parameters*]
    // General way to pass pre-commands to a compiled executable

NC Simulator:
```
% ncvlog_top -s file1.v file2.v specman.v
   // Invokes an executable named ncvlog_top to start Specman Elite
   // with NC Simulator, and load Verilog files file1.v and file2.v
```

NC Simulator:
```
%specsim -p "@batch.ecom"  ncvlog_top -s file1.v file2.v specman.v
  // Same as above, but with optional pre-commands
```

ModelSim:
```
% specsim -p "@batch.ecom" vsim  -keepstdout top < batch.do
```

VCS:
```
% specsim -p "@batch.ecom" vcs_cpu_top -s -i batch.cmd
```

Using a Specman Elite Command File

@*file-name* [*parameter* ...]

```
Example:
    // Contents of my_batch.ecom file:
        load <1>;
        out(“<2> is <3>”);
    Execute my_batch.ecom:
        Specman> @my_batch my_code Today Wednesday
    Result:
        Loads my_code.e, prints Today is Wednesday
```

Record Commands

record start [**-dir =** *dir-name*][**-comment = "***comment-text***"**]

Configuration Commands

config *category* **-***option* **=** *value***;**

Category	Options
print	radix, title, window, raw, items, list_from, list_is_horizontal, list_lines, list_starts_on_right, list_grouping, list_of_bit_in_hex, list_index_radix, list_end_flag, full, source_lines, line_size
cover	at_least_multiplier, grading_formula, show_mode, verbose_interface, sorted, max_int_buckets, absolute_max_buckets, max_gui_buckets, mode, test_name, run_name, tag_name, dir, file_name, show_file_names, show_sub_holes, show_instances_only, show_partial_grade, ranking_cost, ranking_precision, gui_sync_mode, check_illegal_immediately, hole_color, illegal_bucket_color, chart_colors
gen	seed, default_max_list_size, reorder_fields, absolute_max_list_size, max_depth, max_structs, warn, resolve_cycles, check_unsatisfied_cons
gui	auto_scroll
run	tick_max, error_command, exit_on, use_manual_tick
memory	gc_threshold, gc_increment, max_size, absolute_max_size, print_msg, retain_trace_structs
misc	warn, pre_specman_path, post_specman_path, short_is_signed
debug	watch_list_items
wave	working_mode, auto_refresh, register_structs, use_wave, stub_message_len, stub_output, stub_errors, stub_events, event_data, stub_integers, stub_strings, stub_strings_len, stub_booleans, list_items, thread_code_line, hierarchy_name, port, dump_file, timeout

show config [*category* [*option*]]

write config [**to**] *file-name*

read config [**from**] *file-name*

Test Phase Commands

test [*-option* = *value*…]	**setup_test**	**generate** [*-option* = *value*…]
start [*-option* = *value*…]		**run** [*-option* = *value*…]
extract	**check**	**finalize_test**

Test Phase Command Options

seed = *n* **| random**	**default_max_list_size =** *n*
max_depth = *n*	**absolute_max_list_size =** *n*
max_structs = *n*	**warn = TRUE | FALSE**
reorder_fields = TRUE | FALSE	
resolve_cycles = TRUE | FALSE	
check_unsatisfied_cons = TRUE | FALSE	

Saving and Restoring the State

load *file-name …* **reload** [**-nokeep**]

save *file-name*

restore [**-override**] [**-nokeep**] [*file-name*]

Coverage Commands

read cover *file-name* | **-merge -file =** *file-name*

write cover [**-merge**] *file-name*

clear cover

show cover [**-kind** = **full**|**summary**|**spreadsheet**]
[**-file** = *file-name*] [**-contributors**[= *num*]] [**-window**]
[*struct-type*[.*cover-group*[(*inst*)][.*item-name*]]]

show cover -tests

show cover -def [*struct-name*[.*event-name*[.*item-name*]]]

show cover -new -cross = (*struct-type.cover-group.item-name, ...*)
[**-interval** = (*struct-type.event-name*, [*struct-type.event-name* | **next**])]
[**-only_simultaneous**] [**-win**]

show cover -unique_buckets *file_name*

include cover[**_tests**] *full-run-name* [**on**|**off**]

rank cover [**-sort_only**] [**-recover**] [**-window**] [**-file**=*file_name*]
[**-initial_list**=*file_name*] [*item-wild-cards*]

Waveform-Related Commands

set wave [**-mode=***working-mode*] *viewer*

wave [**-when** [**=** *when-regular-expr*]]
[**-field**[**s**] [**=** *fields-regular-expr*]]
[**-event**[**s**] [**-event_data=***event-data*]] [**-thread**[**s**]
[**-code_line=***bool*]] *expr*

wave event [**-data=***data-option*] [*struct-type*.*event-type*]

wave out

Memory Commands

show memory [**-recursive**] [*struct-type* | *unit-type*]

who is [**-full**] *struct-expr* // show paths for all pointers to a struct

Event Commands

collect events [*event-name* [,…]] [**on** | **off**]

echo events [*event-name* [,…]] [**on** | **off**]

delete events

show events [*event-name* | [*num* [..[*num*]]]

show event definitions [*event-name*, …]

show events -chart [*time-value* | **-prev** | **-next** | **-beginning** | **-end**]
[*event-name*, …]

Show Pack and Unpack Commands

show pack(*options*: pack_options, *expr, ...***)**

show unpack(*options*: pack_options, *value-expr, target-expr, ...* **)**

Show Modules Command

show modules

Log Commands

set log *file-name* **set log off**

Shell Commands

shell *shell-command*

Print and Report Commands

Note: **print** and **report** can also be used in **e** code as actions.

print *expr, …* [**using** *print-options*]

report *list-expr*, **{**[*headers*]**}**, *expr,…* [**using** *print-options*]

Note: Use the **show config print** command to display print options.
Examples:

```
print sys.packets using radix=HEX
report sys.packets, {"Addr \t Indx"; "%d \t %d"},.address,index
```

tree [*struct-inst* | *list-expr*] // display the contents of a struct or list

Generation Debugger Commands

col[**lect**] **generation** [**off**]

show gen [**-instance** *inst-name*[.*field-name*]]

Source Code Debugger Commands

continue [**to** *breakpoint-syntax*] **step_anywhere**

step **next** **finish** **abort**

In the next two sections, the *#thread-handle* option can only be used with the "l" (local) form of the command (e.g. **lbreak**, but not **break**). The special events and special wild cards used as options for some of the commands are listed separately at the end.

Setting Breakpoints

[**l**]**break** [**once**] [**on**] **call** [**extension**]
[*struct-wild-card*.]*method-wild-card* [**@***module-name*]
[**#**[*thread-handle*] [**if** *bool-expr*]

[**l**]**break** [**once**] [**on**] [**return**] [**extension**]
[*struct-wild-card*.]*method-wild-card* [**@***module-name*]
[**#**[*thread-handle*]] [**if** *bool-expr*]

[**l**]**break** [**once**] [**on**] **line** [*line-number*] [**@***module-name*]
[**#**[*thread-handle*]] [**if** *bool-expr*]

[**l**]**break** [**once**] [**on**] *special-event-name* [*special-wild-card*]
[**@***module-name*] [**#**[*thread-handle*]] [**if** *bool-expr*]

[**l**]**break** [**once**] [**on**] **event** [[*struct-wild-card*.]*event-wild-card*]
[**@***module-name*] [**#**[*thread-handle*]] [**if** *bool-expr*]

break [**once**] [**on**] **change** *expr*

break [**once**] [**on**] **error**

break [**once**] [**on**] **interrupt**

break [**once**] [**on**] **simulator**

break [**on**] **alloc** [*memory-size*]

Managing Breakpoints

delete break [**last** | *id-number* | **"***pattern***"**]

disable break [**last** | *id-number* | **"***pattern***"**]

enable break [**last** | *id-number* | **"***pattern***"**]

show breakpoint

Setting and Managing Watches

[**l**]**watch** *expr* [**-radix = DEC**|**HEX**|**BIN**] [**-items =** *value*] [**#***thread-id*]

update watch *watch-id* [**radix = DEC**|**HEX**|**BIN**]
[**-items =** *value*|**default**]

show watch **delete watch** [*watch-id*]

Setting Traces

[**l**]**trace** [**once**] [**on**] **call** [**extension**] [*struct-wild-card*.]*method-wild-card* [**@***module-name*] [**#**[*thread-handle*]] [**if** *bool-expr*]

[**l**]**trace** [**once**] [**on**] **return** [**extension**] [*struct-wild-card*.]*method-wild-card* [**@***module-name*] [**#**[*thread-handle*]] [**if** *bool-expr*]

[**l**]**trace** [**once**] [**on**] **line** [*line-number*] [**@***module-name*] [**if** *bool-expr*]

[**l**]**trace** [**once**] [**on**] *special-event-name* [*special-wild-card*]
[**@***module-name*] [**#**[*thread-handle*]] [**if** *bool-expr*]

trace [**once**] [**on**] **change** *expr*

trace [**on**] **packing** **trace** [**on**] **reparse**

Special Events and Special Wild Cards

Special Event Name	Special Wild Card
tcm_start	*struct-wild-card.tcm-wild-card*
tcm_end	*struct-wild-card.tcm-wild-card*
tcm_call	*struct-wild-card.tcm-wild-card*
tcm_return	*struct-wild-card.tcm-wild-card*
tcm_wait	*struct-wild-card.tcm-wild-card*
tcm_state	*struct-wild-card.tcm-wild-card*
call	*struct-wild-card.method-wild-card*
return	*struct-wild-card.method-wild-card*
sim_read	*signal-name-wild-card*
sim_write	*signal-name-wild-card*
output	*text wild-card*

Command-Line Mode Debugging Commands

show stack // show the calls stack for the current thread

show threads // show all threads

show thread source // show the **e** source for the current thread

show thread tree // show the full tree of calls for the current thread

show thread *#thread-handle*