ModelSim/VHDL, ModelSim/VLOG, ModelSim/LNL, and ModelSim/PLUS are produced by Model Technology™ Incorporated. Unauthorized copying, duplication, or other reproduction is prohibited without the written consent of Model Technology.

The information in this manual is subject to change without notice and does not represent a commitment on the part of Model Technology. The program described in this manual is furnished under a license agreement and may not be used or copied except in accordance with the terms of the agreement. The online documentation provided with this product may be printed by the end-user. The number of copies that may be printed is limited to the number of licenses purchased.

ModelSim is a registered trademark and Signal Spy, TraceX, and ChaseX are trademarks of Model Technology Incorporated. Model Technology is a trademark of Mentor Graphics Corporation. PostScript is a registered trademark of Adobe Systems Incorporated. UNIX is a registered trademark of AT&T in the USA and other countries. FLEXlm is a trademark of Globetrotter Software, Inc. IBM, AT, and PC are registered trademarks, AIX and RISC System/6000 are trademarks of International Business Machines Corporation. Windows, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation. OSF/Motif is a trademark of the Open Software Foundation, Inc. in the USA and other countries. SPARC is a registered trademark and SPARCstation is a trademark of SPARC International, Inc. Sun Microsystems is a registered trademark, and Sun, SunOS and OpenWindows are trademarks of Sun Microsystems, Inc. All other trademarks and registered trademarks are the properties of their respective holders.

Copyright (c) 1990 -2002, Model Technology Incorporated. All rights reserved. Confidential. Online documentation may be printed by licensed customers of Model Technology Incorporated for internal business purposes only.

Model Technology Incorporated
10450 SW Nimbus Avenue / Bldg. R-B
Portland OR 97223-4347 USA

phone: 503-641-1340
fax: 503-526-5410
e-mail: support@model.com, sales@model.com
home page: http://www.model.com
support page: http://www.model.com/support
Technical support and updates

The Model Technology web site includes links to support, software updates, and many other information sources.

Support

www.model.com/support/default.asp

Customers in Europe should contact their distributor for support. See www.model.com/contact_us.asp for distributor contact information.

Updates

www.model.com/products/release.asp

Latest version email

Place your name on our list for email notification of news and updates. www.model.com/support/register_news_list.asp
# Table of Contents

### Syntax and conventions (CR-9)
- Documentation conventions CR-10
- Command return values CR-11
- Command shortcuts CR-11
- Command history shortcuts CR-11
- Numbering conventions CR-12
- File and directory pathnames CR-13
- HDL item pathnames CR-14
- Wildcard characters CR-16
- ModelSim variables CR-17
- Simulation time units CR-17
- Comments in argument files CR-17
- GUI_expression_format CR-18

### Commands (CR-27)
- Command reference table CR-28
- .main clear CR-37
- .wave.tree interrupt CR-38
- .wave.tree zoomfull CR-39
- .wave.tree zoomin CR-40
- .wave.tree zoomlast CR-41
- .wave.tree zoomout CR-42
- .wave.tree zoomrange CR-43
- abort CR-44
- add button CR-45
- add dataflow CR-47
- add list CR-48
- add_menu CR-51
- add_menucb CR-53
- add_menuitem CR-54
- add_separator CR-55
- add_submenu CR-56
- add wave CR-57
- alias CR-61
- batch_mode CR-62
- bd CR-63
- bookmark add wave CR-64
- bookmark delete wave CR-65
- bookmark goto wave CR-66
- bookmark list wave CR-67
- bp CR-68
- cd CR-71
- change CR-72
- change_menu_cmd CR-73
- check contention add CR-74
- check contention config CR-75
- check contention off CR-76
- check float add CR-77
- check float config CR-78
- check float off CR-79
- check stable off CR-80
- check stable on CR-81
- checkpoint CR-82
- compare add CR-83
- compare annotate CR-86
- compare clock CR-87
- compare configure CR-89
- compare continue CR-90
- compare delete CR-91
- compare end CR-92
- compare info CR-93
- compare list CR-95
- compare options CR-96
- compare reload CR-99
- compare reset CR-100
- compare run CR-101
- compare savediffs CR-102
- compare saverules CR-103
- compare see CR-104
- compare start CR-106
- compare stop CR-108
- compare update CR-109
- configure CR-110
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>CR-114</td>
</tr>
<tr>
<td>coverage clear</td>
<td>CR-116</td>
</tr>
<tr>
<td>coverage exclude clear</td>
<td>CR-117</td>
</tr>
<tr>
<td>coverage exclude disable</td>
<td>CR-118</td>
</tr>
<tr>
<td>coverage exclude enable</td>
<td>CR-119</td>
</tr>
<tr>
<td>coverage exclude load</td>
<td>CR-120</td>
</tr>
<tr>
<td>coverage reload</td>
<td>CR-121</td>
</tr>
<tr>
<td>coverage report</td>
<td>CR-122</td>
</tr>
<tr>
<td>dataset alias</td>
<td>CR-123</td>
</tr>
<tr>
<td>dataset clear</td>
<td>CR-124</td>
</tr>
<tr>
<td>dataset close</td>
<td>CR-125</td>
</tr>
<tr>
<td>dataset info</td>
<td>CR-126</td>
</tr>
<tr>
<td>dataset list</td>
<td>CR-127</td>
</tr>
<tr>
<td>dataset open</td>
<td>CR-128</td>
</tr>
<tr>
<td>dataset rename</td>
<td>CR-129</td>
</tr>
<tr>
<td>dataset save</td>
<td>CR-130</td>
</tr>
<tr>
<td>dataset snapshot</td>
<td>CR-131</td>
</tr>
<tr>
<td>delete</td>
<td>CR-133</td>
</tr>
<tr>
<td>describe</td>
<td>CR-134</td>
</tr>
<tr>
<td>disablebp</td>
<td>CR-135</td>
</tr>
<tr>
<td>disable_menu</td>
<td>CR-136</td>
</tr>
<tr>
<td>disable_menuitem</td>
<td>CR-137</td>
</tr>
<tr>
<td>do</td>
<td>CR-138</td>
</tr>
<tr>
<td>down</td>
<td>CR-139</td>
</tr>
<tr>
<td>drivers</td>
<td>CR-141</td>
</tr>
<tr>
<td>dumplog64</td>
<td>CR-142</td>
</tr>
<tr>
<td>echo</td>
<td>CR-143</td>
</tr>
<tr>
<td>edit</td>
<td>CR-144</td>
</tr>
<tr>
<td>enablebp</td>
<td>CR-145</td>
</tr>
<tr>
<td>enable_menu</td>
<td>CR-146</td>
</tr>
<tr>
<td>enable_menuitem</td>
<td>CR-147</td>
</tr>
<tr>
<td>environment</td>
<td>CR-148</td>
</tr>
<tr>
<td>examine</td>
<td>CR-149</td>
</tr>
<tr>
<td>exit</td>
<td>CR-152</td>
</tr>
<tr>
<td>find</td>
<td>CR-153</td>
</tr>
<tr>
<td>force</td>
<td>CR-156</td>
</tr>
<tr>
<td>getactivecursortime</td>
<td>CR-159</td>
</tr>
<tr>
<td>getactivemarkertime</td>
<td>CR-160</td>
</tr>
<tr>
<td>help</td>
<td>CR-161</td>
</tr>
<tr>
<td>history</td>
<td>CR-162</td>
</tr>
<tr>
<td>lecho</td>
<td>CR-163</td>
</tr>
<tr>
<td>left</td>
<td>CR-164</td>
</tr>
<tr>
<td>log</td>
<td>CR-166</td>
</tr>
<tr>
<td>lshift</td>
<td>CR-168</td>
</tr>
<tr>
<td>lsublist</td>
<td>CR-169</td>
</tr>
<tr>
<td>macro_option</td>
<td>CR-170</td>
</tr>
<tr>
<td>modelsim</td>
<td>CR-171</td>
</tr>
<tr>
<td>next</td>
<td>CR-172</td>
</tr>
<tr>
<td>noforce</td>
<td>CR-173</td>
</tr>
<tr>
<td>nolog</td>
<td>CR-174</td>
</tr>
<tr>
<td>notepad</td>
<td>CR-176</td>
</tr>
<tr>
<td>noview</td>
<td>CR-177</td>
</tr>
<tr>
<td>nowhen</td>
<td>CR-178</td>
</tr>
<tr>
<td>onbreak</td>
<td>CR-179</td>
</tr>
<tr>
<td>onElabError</td>
<td>CR-180</td>
</tr>
<tr>
<td>onerror</td>
<td>CR-181</td>
</tr>
<tr>
<td>pause</td>
<td>CR-182</td>
</tr>
<tr>
<td>play</td>
<td>CR-183</td>
</tr>
<tr>
<td>power add</td>
<td>CR-184</td>
</tr>
<tr>
<td>power report</td>
<td>CR-185</td>
</tr>
<tr>
<td>power reset</td>
<td>CR-186</td>
</tr>
<tr>
<td>printenv</td>
<td>CR-187</td>
</tr>
<tr>
<td>profile clear</td>
<td>CR-188</td>
</tr>
<tr>
<td>profile interval</td>
<td>CR-189</td>
</tr>
<tr>
<td>profile off</td>
<td>CR-190</td>
</tr>
<tr>
<td>profile on</td>
<td>CR-191</td>
</tr>
<tr>
<td>profile option</td>
<td>CR-192</td>
</tr>
<tr>
<td>profile report</td>
<td>CR-193</td>
</tr>
<tr>
<td>project</td>
<td>CR-194</td>
</tr>
<tr>
<td>property list</td>
<td>CR-195</td>
</tr>
<tr>
<td>property wave</td>
<td>CR-196</td>
</tr>
<tr>
<td>pwd</td>
<td>CR-197</td>
</tr>
<tr>
<td>quietly</td>
<td>CR-198</td>
</tr>
<tr>
<td>quit</td>
<td>CR-199</td>
</tr>
<tr>
<td>radix</td>
<td>CR-200</td>
</tr>
<tr>
<td>record</td>
<td>CR-201</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
</tr>
<tr>
<td>report CR-202</td>
<td></td>
</tr>
<tr>
<td>restart CR-204</td>
<td></td>
</tr>
<tr>
<td>restore CR-206</td>
<td></td>
</tr>
<tr>
<td>resume CR-207</td>
<td></td>
</tr>
<tr>
<td>right CR-208</td>
<td></td>
</tr>
<tr>
<td>run CR-210</td>
<td></td>
</tr>
<tr>
<td>search CR-212</td>
<td></td>
</tr>
<tr>
<td>searchlog CR-214</td>
<td></td>
</tr>
<tr>
<td>seetime CR-216</td>
<td></td>
</tr>
<tr>
<td>shift CR-217</td>
<td></td>
</tr>
<tr>
<td>show CR-218</td>
<td></td>
</tr>
<tr>
<td>simstats CR-219</td>
<td></td>
</tr>
<tr>
<td>splitio CR-220</td>
<td></td>
</tr>
<tr>
<td>status CR-221</td>
<td></td>
</tr>
<tr>
<td>step CR-222</td>
<td></td>
</tr>
<tr>
<td>stop CR-223</td>
<td></td>
</tr>
<tr>
<td>tb CR-224</td>
<td></td>
</tr>
<tr>
<td>toggle add CR-225</td>
<td></td>
</tr>
<tr>
<td>toggle report CR-226</td>
<td></td>
</tr>
<tr>
<td>toggle reset CR-227</td>
<td></td>
</tr>
<tr>
<td>transcribe CR-228</td>
<td></td>
</tr>
<tr>
<td>transcript CR-229</td>
<td></td>
</tr>
<tr>
<td>tssi2mti CR-230</td>
<td></td>
</tr>
<tr>
<td>up CR-231</td>
<td></td>
</tr>
<tr>
<td>vcd add CR-233</td>
<td></td>
</tr>
<tr>
<td>vcd checkpoint CR-234</td>
<td></td>
</tr>
<tr>
<td>vcd comment CR-235</td>
<td></td>
</tr>
<tr>
<td>vcd dumpports CR-236</td>
<td></td>
</tr>
<tr>
<td>vcd dumpportsall CR-238</td>
<td></td>
</tr>
<tr>
<td>vcd dumpportsflush CR-239</td>
<td></td>
</tr>
<tr>
<td>vcd dumpportslimit CR-240</td>
<td></td>
</tr>
<tr>
<td>vcd dumpportsoff CR-241</td>
<td></td>
</tr>
<tr>
<td>vcd dumpportson CR-242</td>
<td></td>
</tr>
<tr>
<td>vcd file CR-243</td>
<td></td>
</tr>
<tr>
<td>vcd files CR-245</td>
<td></td>
</tr>
<tr>
<td>vcd flush CR-247</td>
<td></td>
</tr>
<tr>
<td>vcd limit CR-248</td>
<td></td>
</tr>
<tr>
<td>vcd off CR-249</td>
<td></td>
</tr>
<tr>
<td>vcd on CR-250</td>
<td></td>
</tr>
<tr>
<td>vcd2wlf CR-251</td>
<td></td>
</tr>
<tr>
<td>vcom CR-252</td>
<td></td>
</tr>
<tr>
<td>vdel CR-258</td>
<td></td>
</tr>
<tr>
<td>vdir CR-259</td>
<td></td>
</tr>
<tr>
<td>verror CR-260</td>
<td></td>
</tr>
<tr>
<td>vgencomp CR-261</td>
<td></td>
</tr>
<tr>
<td>view CR-263</td>
<td></td>
</tr>
<tr>
<td>virtual count CR-265</td>
<td></td>
</tr>
<tr>
<td>virtual define CR-266</td>
<td></td>
</tr>
<tr>
<td>virtual delete CR-267</td>
<td></td>
</tr>
<tr>
<td>virtual describe CR-268</td>
<td></td>
</tr>
<tr>
<td>virtual expand CR-269</td>
<td></td>
</tr>
<tr>
<td>virtual function CR-270</td>
<td></td>
</tr>
<tr>
<td>virtual hide CR-273</td>
<td></td>
</tr>
<tr>
<td>virtual log CR-274</td>
<td></td>
</tr>
<tr>
<td>virtual nohide CR-276</td>
<td></td>
</tr>
<tr>
<td>virtual nolog CR-277</td>
<td></td>
</tr>
<tr>
<td>virtual region CR-279</td>
<td></td>
</tr>
<tr>
<td>virtual save CR-280</td>
<td></td>
</tr>
<tr>
<td>virtual show CR-281</td>
<td></td>
</tr>
<tr>
<td>virtual signal CR-282</td>
<td></td>
</tr>
<tr>
<td>virtual type CR-285</td>
<td></td>
</tr>
<tr>
<td>vlib CR-287</td>
<td></td>
</tr>
<tr>
<td>vlog CR-288</td>
<td></td>
</tr>
<tr>
<td>vmake CR-296</td>
<td></td>
</tr>
<tr>
<td>vmap CR-297</td>
<td></td>
</tr>
<tr>
<td>vsim CR-298</td>
<td></td>
</tr>
<tr>
<td>vsim&lt;info&gt; CR-312</td>
<td></td>
</tr>
<tr>
<td>vsrcsource CR-313</td>
<td></td>
</tr>
<tr>
<td>when CR-314</td>
<td></td>
</tr>
<tr>
<td>where CR-318</td>
<td></td>
</tr>
<tr>
<td>wlf2log CR-319</td>
<td></td>
</tr>
<tr>
<td>wlfrecover CR-321</td>
<td></td>
</tr>
<tr>
<td>write cell_report CR-322</td>
<td></td>
</tr>
<tr>
<td>write format CR-323</td>
<td></td>
</tr>
<tr>
<td>write list CR-325</td>
<td></td>
</tr>
<tr>
<td>write preferences CR-326</td>
<td></td>
</tr>
</tbody>
</table>
write report CR-327
write transcript CR-328
write tssi CR-329
write wave CR-331

Licensing Agreement (CR-333)

Index (CR-337)
# Syntax and conventions

## Chapter contents

- Documentation conventions ........................................ CR-10
- Command return values .............................................. CR-11
- Command shortcuts .................................................. CR-11
- Command history shortcuts ......................................... CR-11
- Numbering conventions ............................................. CR-12
- File and directory pathnames ...................................... CR-13
- HDL item pathnames .................................................. CR-14
- Wildcard characters .................................................. CR-16
- ModelSim variables ................................................... CR-17
- Simulation time units ................................................ CR-17
- Comments in argument files ....................................... CR-17
- GUI_expression_format .............................................. CR-18
Documentation conventions

This manual uses the following conventions to define ModelSim command syntax.

<table>
<thead>
<tr>
<th>Syntax notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; &gt;</td>
<td>angled brackets surrounding a syntax item indicate a user-defined argument; do not enter the brackets in commands</td>
</tr>
<tr>
<td>[ ]</td>
<td>square brackets generally indicate an optional item; if the brackets surround several words, all must be entered as a group; the brackets are not entered(^a)</td>
</tr>
<tr>
<td>{ }</td>
<td>braces indicate that the enclosed expression contains one or more spaces yet should be treated as a single argument, or that the expression contains square brackets for an index; for either situation, the braces are entered</td>
</tr>
<tr>
<td>...</td>
<td>an ellipsis indicates items that may appear more than once; the ellipsis itself does not appear in commands</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>monospaced type</td>
<td>monospaced type is used in command examples</td>
</tr>
<tr>
<td>#</td>
<td>comments included with commands are preceded by the number sign (#); useful for adding comments to DO files (macros)</td>
</tr>
</tbody>
</table>

\(^a\) One exception to this rule is when you are using Verilog syntax to designate an array slice. For example,

\[
\text{add wave \{vector1[4:0]\}}
\]

The square brackets in this case denote an index. The braces prevent the Tcl interpreter from treating the text within the square brackets as a Tcl command.

\[\textbf{Note:} \] Neither the prompt at the beginning of a line nor the <Enter> or <Return> key that ends a line is shown in the command examples.
Command return values

All simulator commands are invoked using Tcl. For most commands that write information
to the Main window, that information is also available as a Tcl result. By using command
substitution the results can be made available to another command or assigned to a Tcl
variable. For example:

```tcl
set aluinputs [find -in alu/*]
```

sets variable "aluinputs" to the result of the `find` command (CR-153).

Command shortcuts

You may abbreviate command syntax, but there’s a catch — the minimum number of
characters required to execute a command are those that make it unique. Remember, as we
add new commands some of the old shortcuts may not work. For this reason ModelSim
does not allow command name abbreviations in macro files. This minimizes your need to
update macro files as new commands are added.

Command history shortcuts

The simulator command history may be reviewed, or commands may be reused, with these
shortcuts at the ModelSim/VSIM prompt:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!!</td>
<td>repeats the last command</td>
</tr>
<tr>
<td>!n</td>
<td>repeats command number n; n is the VSIM prompt number (e.g., for this prompt: VSIM 12&gt;, n =12)</td>
</tr>
<tr>
<td>!abc</td>
<td>repeats the most recent command starting with &quot;abc&quot;</td>
</tr>
<tr>
<td><code>xyz</code>^ab^</td>
<td>replaces &quot;xyz&quot; in the last command with &quot;ab&quot;</td>
</tr>
<tr>
<td>up and down arrows</td>
<td>scrolls through the command history with the keyboard arrows</td>
</tr>
<tr>
<td>click on prompt</td>
<td>left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor</td>
</tr>
<tr>
<td>his or history</td>
<td>shows the last few commands (up to 50 are kept)</td>
</tr>
</tbody>
</table>
Numbering conventions

Numbers in ModelSim can be expressed in either VHDL or Verilog style. Two styles can be used for VHDL numbers, one for Verilog.

VHDL numbering conventions

The first of two VHDL number styles is:

\[-\] [ radix # ] value [ # ]

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>indicates a negative number; optional</td>
</tr>
<tr>
<td>radix</td>
<td>can be any base in the range 2 through 16 (2, 8, 10, or 16); by default, numbers are assumed to be decimal; optional</td>
</tr>
<tr>
<td>value</td>
<td>specifies the numeric value, expressed in the specified radix; required</td>
</tr>
<tr>
<td>#</td>
<td>is a delimiter between the radix and the value; the first # sign is required if a radix is used, the second is always optional</td>
</tr>
</tbody>
</table>

Note: A `-` can also be used to designate a "don't care" element when you search for a signal in the List or Wave window. If you want the `-` to be read as a "don't care" element, rather than a negative sign, be sure to enclose the number in double quotes. For instance, you would type "-0110--" as opposed to -0110--. If you don’t include the double quotes, ModelSim will read the `-` as a negative sign.

Examples

16#FFca23#
2#11111110
-23749

The second VHDL number style is:

base "value"

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>specifies the base; binary: B, octal: O, hex: X; required</td>
</tr>
<tr>
<td>value</td>
<td>specifies digits in the appropriate base with optional underscore separators; default is decimal; required</td>
</tr>
</tbody>
</table>

Examples

B"11111110"
X"FFca23"
Verilog numbering conventions

Verilog numbers are expressed in the style:

\[-\] [ size ] [ base ] value

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>indicates a negative number; optional</td>
</tr>
<tr>
<td>size</td>
<td>the number of bits in the number; optional</td>
</tr>
<tr>
<td>base</td>
<td>specifies the base; binary: ‘b or ‘B, octal: ‘o or ‘O, decimal: ‘d or ‘D, hex: ‘h or ‘H; optional</td>
</tr>
<tr>
<td>value</td>
<td>specifies digits in the appropriate base with optional underscore separators; default is decimal, required</td>
</tr>
</tbody>
</table>

▶ Note: A ‘-’ can also be used to designate a "don't care" element when you search for a signal in the List or Wave windows. If you want the ‘-’ to be read as a "don’t care" element, rather than a negative sign, be sure to enclose the number in double quotes. For instance, you would type "-0110--" as opposed to 7'b-0110--. If you don’t include the double quotes, ModelSim will read the ‘-’ as a negative sign.

Examples

- b11111110 8'b11111110
- 'Hffca23 21'H1fca23
- -23749

File and directory pathnames

Several ModelSim commands have arguments that point to files or directories. For example, the -y argument to vlog specifies the Verilog source library directory to search for undefined modules. Spaces in file pathnames must be escaped or the entire path must be enclosed in quotes. For example:

vlog top.v -y C:/Documents\ and\ Settings/mcarnes/simprims

or

vlog top.v -y "C:/Documents and Settings/mcarnes/simprims"
HDL item pathnames

VHDL and Verilog items are organized hierarchically. Each of the following HDL items creates a new level in the hierarchy:

- **VHDL**
  - component instantiation statement, block statement, and package
- **Verilog**
  - module instantiation, named fork, named begin, task and function

Each level in the hierarchy is also known as a "region."

Multiple levels in a pathname

Multiple levels in a pathname are separated by the character specified in the PathSeparator variable. The default is "/", but it can be set to any single character, such as "." for Verilog naming conventions, or ":" for VHDL IEEE 1076-1993 naming conventions. See the PathSeparator (UM-449) variable for more information.

Absolute pathnames

Absolute pathnames begin with the path separator character. The first name in the path should be the name of a top-level entity or module, but if you leave it off then the first top-level entity or module will be assumed. VHDL designs only have one top level, so it doesn't matter if it is included in the pathname. For example, if you are referring to the signal `clk` in the top-level entity named `top`, then both of the following pathnames are correct:

```
/top/clk
/clk
```

► **Note:** Since Verilog designs may contain multiple top-level modules, a path name may be ambiguous if you leave off the top-level module name.

Relative pathnames

Relative pathnames do not start with the path separator, and are relative to the current environment. The current environment defaults to the first top-level entity or module and may be changed by the environment command or by clicking on hierarchy levels in the Structure window. Each new level in the pathname is first searched downwards relative to the current environment, but if not found is then searched for upwards (same search rules used in Verilog hierarchical names).

Environment variables and pathnames

You can substitute environment variables for pathnames in any argument that requires a pathname. For example:

```
vlog -v $lib_path/und1
```

Assuming you have defined `$lib_path` on your system, vlog will locate the source library file `und1` and search it for undefined modules. See "Environment variables" (UM-441) for more information.
Indexing signals, memories, and nets

VHDL array signals, and Verilog memories and vector nets can be sliced or indexed. Indexes must be numeric, since the simulator does not know the actual index types. Slice ranges may be represented in either VHDL or Verilog syntax, irrespective of the setting of the PathSeparator (UM-449).

Name case sensitivity

Name case sensitivity is different for VHDL and Verilog. VHDL names are not case sensitive except for extended identifiers in VHDL 1076-1993. In contrast, all Verilog names are case sensitive. Names in ModelSim commands are case sensitive when matched against case sensitive identifiers, otherwise they are not case sensitive.

Extended identifiers

The following are supported formats for extended identifiers for any command that takes an identifier.

```plaintext
{\ext ident!}     # Note trailing space.
\\ext\ ident\!\\    # All non-alpha characters escaped
```

Naming fields in VHDL signals

Fields in VHDL record signals can be specified using the form:

```plaintext
signal_name.field_name
```
Example pathnames

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>specifies the item clk in the current environment</td>
</tr>
<tr>
<td>/top/clk</td>
<td>specifies the item clk in the top-level design unit.</td>
</tr>
<tr>
<td>/top/block1/u2/clk</td>
<td>specifies the item clk, two levels down from the</td>
</tr>
<tr>
<td></td>
<td>top-level design unit</td>
</tr>
<tr>
<td>block1/u2/clk</td>
<td>specifies the item clk, two levels down from the</td>
</tr>
<tr>
<td></td>
<td>current environment</td>
</tr>
<tr>
<td>array_sig(4)</td>
<td>specifies an index of an array item</td>
</tr>
<tr>
<td>{array_sig(1 to 10)}</td>
<td>specifies a slice of an array item in VHDL syntax</td>
</tr>
<tr>
<td>{mysignal[31:0]}</td>
<td>specifies a slice of an array item in Verilog syntax</td>
</tr>
<tr>
<td>record_sig.field</td>
<td>specifies a field of a record</td>
</tr>
</tbody>
</table>

Wildcard characters

Wildcard characters can be used in HDL item names in some simulator commands. Conventions for wildcards are as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches any sequence of characters</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
</tbody>
</table>

The WildcardFilter Tcl preference variable filters matching items for the add wave, add log, add list, and find commands.
**ModelSim variables**

Several variables are available to control simulation, provide simulator state feedback, or modify the appearance of the ModelSim GUI. To take effect, some variables, such as environment variables, must be set prior to simulation.

ModelSim variables can be referenced in simulator commands by preceding the name of the variable with the dollar sign ($) character. ModelSim uses global Tcl variables for simulator state variables, simulator control variables, simulator preference variables and user-defined variables (See [http://www.model.com/resources/pref_variables/frameset.htm](http://www.model.com/resources/pref_variables/frameset.htm) for a list of Tcl preference variables.)

See Appendix A - ModelSim variables in the User’s Manual for more information on variables.

**Variable settings report**

The report command (CR-202) returns a list of current settings for either the simulator state, or simulator control variables.

**Simulation time units**

You can specify the time unit for delays in all simulator commands that have time arguments:

```
force clk 1 50 ns, 1 100 ns -repeat 1 us
run 2 ms
```

Note that all the time units in a ModelSim command need not be the same.

Unless you specify otherwise as in the examples above, simulation time is always expressed using the resolution units that are specified by the UserTimeUnit variable. See UserTimeUnit (UM-450).

By default, the specified time units are assumed to be relative to the current time unless the value is preceded by the character @, which signifies an absolute time specification.

**Comments in argument files**

Argument files may be loaded with the -f <filename> argument of the vcom, vlog, and vsim commands. The -f <filename> argument specifies a file that contains more command line arguments.

Comments within the argument files follow these rules:

- All text in a line beginning with // to its end is treated as a comment.
- All text bracketed by /* ... */ is treated as a comment.

Also, program arguments can be placed on separate lines in the argument file, with the newline characters treated as space characters. There is no need to put \ at the end of each line.
GUI_expression_format

The GUI_expression_format is an option of several simulator commands that operate within ModelSim’s GUI environment. The expressions help you locate and examine HDL items within the List and Wave windows (expressions may also be used through the Edit > Search menu in both windows). The commands that use the expression format are:

- compare add (CR-83), compare clock (CR-87), compare configure (CR-89), configure (CR-110),
- down (CR-139), examine (CR-149), left (CR-164), right (CR-208), searchlog (CR-214),
- up (CR-231), virtual function (CR-270), and virtual signal (CR-282)

Expressions may be typed directly on the VSIM command line, or you can use the "The GUI Expression Builder" (UM-305).

Expression typing

GUI expressions are typed. The supported types consist of six scalar types and two array types.

Scalar types

The scalar types are as follows: boolean, integer, real (64-bit integer), enumeration, and signal state. Signal states are represented by the nine VHDL std_logic states: 'U' 'X' '0' '1' 'Z' 'H' 'L' 'W' and ' -'. Verilog states 0, 1, x, and z are mapped into these states and the Verilog strengths are ignored. Conversion is done automatically when referencing Verilog nets or registers.

Array types

The array types supported are signed and unsigned arrays of signal states. This would correspond to the VHDL std_logic_array type. Verilog registers are automatically converted to these array types. The array type can be treated as either UNSIGNED or SIGNED, as in the IEEE std_logic_arith package. Normally, referencing a signal array causes it to be treated as UNSIGNED by the expression evaluator; to cause it to be treated as SIGNED, use casting as described below. Numeric operations supported on arrays are performed by the expression evaluator via ModelSim’s built-in numeric_standard (and similar) package routines. The expression evaluator selects the appropriate numeric routine based on SIGNED or UNSIGNED properties of the array arguments and the result.

The enumeration types supported are any VHDL enumerated type. Enumeration literals may be used in the expression as long as some variable of that enumeration type is referenced in the expression. This is useful for sub-expressions of the form:

```
(/memory/state == reading)
```
Signal and subelement naming conventions

ModelSim supports naming conventions for VHDL and Verilog signal pathnames, VHDL array indexing, Verilog bit selection, VHDL subrange specification, and Verilog part selection.

Examples in Verilog and VHDL syntax:

- top.chip.vlogsig
- /top/chip/vhdlssig
- vlogsig[3]
- vhdsig(9)
- vlogsig[5:2]
- vhdsig(5 downto 2)

Concatenation of signals or subelements

Elements in the concatenation that are arrays are expanded so that each element in the array becomes a top-level element of the concatenation. But for elements in the concatenation that are records, the entire record becomes one top-level element in the result. To specify that the records be broken down so that their subelements become top-level elements in the concatenation, use the `concat_flatten` directive. Currently we do not support leaving full arrays as elements in the result. (Please let us know if you need that option.)

If the elements being concatenated are of incompatible base type, a VHDL-style record will be created. The record object can be expanded in the Signals and Wave windows just like an array of compatible type elements.

**Concatenation syntax for VHDL**

\[ <\text{signalOrSliceName}_1> \& <\text{signalOrSliceName}_2> \& \ldots \]

**Concatenation syntax for Verilog**

\[
\begin{align*}
&\{<\text{signalOrSliceName}_1>, <\text{signalOrSliceName}_2>, \ldots \} \\
&\{\text{count}\}<\text{signalOrSliceName}_1>, <\text{signalOrSliceName}_2>, \ldots \}
\end{align*}
\]

Note that the concatenation syntax begins with "\&{" rather than just ".\". Repetition multipliers are supported, as illustrated in the second line. The repetition element itself may be an arbitrary concatenation subexpression.

**Concatenation directives**

The concatenation directive (as illustrated below) can be used to constrain the resulting array range of a concatenation or influence how compound objects are treated. By default, the concatenation will be created with descending index range from \((n-1)\) downto 0, where \(n\) is the number of elements in the array. The `concat_range` directive completely specifies the index range. The `concat_ascending` directive specifies that the index start at zero and increment upwards. The `concat_flatten` directive flattens the signal structure hierarchy. The `concat_sort_wild_ascending` directive gathers signals by name in ascending order (the default is descending).

- `(concat_range 31:0)<concatenationExpr>` # Verilog syntax
- `(concat_range {31:0})<concatenationExpr>` # Also Verilog syntax
- `(concat_range {31 downto 0})<concatenationExpr>` # VHDL syntax
- `(concat_ascending) <concatenationExpr>`
- `(concat_flatten) <concatenationExpr>` # no hierarchy
- `(concat_sort_wild_ascending) <concatenationExpr>`
Examples

&{ "mybusbasename**" }
Gathers all signals in the current context whose names begin with "mybusbasename", sorts those names in descending order, and creates a bus with index range \((n-1)\) downto \(0\), where \(n\) is the number of matching signals found. (Note that it currently does not derive the index name from the tail of the one-bit signal name.)

{concat_range 13:4}&{ "mybusbasename**" }
Specifies the index range to be 13 downto 4, with the signals gathered by name in descending order.

{concat_ascending}&{ "mybusbasename**" }
Specifies an ascending range of 0 to \(n-1\), with the signals gathered by name in descending order.

{concat_ascending}{{concat_sort_wild_ascending}&{"mybusbasename**" }}
Specifies an ascending range of 0 to \(n-1\), with the signals gathered by name in ascending order.

VHDL record field support

Arbitrarily-nested arrays and records are supported, but operators will only operate on one field at a time. That is, the expression \(\{a == b\}\) where \(a\) and \(b\) are records with multiple fields, is not supported. This would have to be expressed as:

\(\{(a.f1 == b.f1) && (a.f2 == b.f2)\ldots\}\)

Examples:

\(\text{vhdlsig}.\text{field1}\)
\(\text{vhdlsig}.\text{field1}.\text{subfield1}\)
\(\text{vhdlsig}.(5).\text{field3}\)
\(\text{vhdlsig}.\text{field4}(3\text{ downto }0)\)

Grouping and precedence

Operator precedence generally follows that of the C language, but we recommend liberal use of parentheses.
### Searching for binary signal values in the GUI

When you use the GUI to search for signal values displayed in 4-state binary radix, you should be aware of how ModelSim maps between binary radix and std_logic. The issue arises because there is no "uninitialized" value in binary, while there is in std_logic (designated by the letter "U"). Consequently, ModelSim relies on mapping tables to determine whether a match occurs between the displayed binary signal value and the underlying std_logic value.

For comparing VHDL std_logic/std_ulogic objects, ModelSim uses the table shown below. An entry of "0" in the table is "no match"; an entry of "1" is a "match"; an entry of "2" is a match only if you set the Tcl variable `STDLOGIC_X_MatchesAnything` to 1. Note that X will match a U, and - will match anything.

<table>
<thead>
<tr>
<th>Search Entry</th>
<th>Matches as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

For comparing Verilog net values, ModelSim uses the table shown below. An entry of "2" is a match only if you set the Tcl variable "VLOG_X_MatchesAnything" to 1.

<table>
<thead>
<tr>
<th>Search Entry</th>
<th>Matches as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
</tr>
</tbody>
</table>

▶ **Note:** This matching algorithm applies only to searching via the GUI; it does not apply to VHDL or Verilog testbenches.
Expression syntax

GUI expressions generally follow C-language syntax, with both VHDL-specific and Verilog-specific conventions supported. These expressions are not parsed by the Tcl parser, and so do not support general Tcl; parentheses should be used rather than curly braces. Procedure calls are not supported.

A GUI expression can include the following elements: Tcl macros, constants, array constants, variables, array variables, signal attributes, operators and casting.

Tcl macros

Macros are useful for pre-defined constants or for entire expressions that have been previously saved. The substitution is done only once, when the expression is first parsed. Macro syntax is:

```
$<name>
```

Substitutes the string value of the Tcl global variable `<name>`.

Constants

<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean value</td>
<td>true false TRUE FALSE</td>
</tr>
<tr>
<td>integer</td>
<td>[0-9]+</td>
</tr>
<tr>
<td>real number</td>
<td><code>&lt;int&gt;</code></td>
</tr>
<tr>
<td>time</td>
<td>integer or real optionally followed by time unit</td>
</tr>
<tr>
<td>enumeration</td>
<td>VHDL user-defined enumeration literal</td>
</tr>
<tr>
<td>single bit constants</td>
<td>expressed as any of the following: 0 1 x Z U H L W ’U’ ’X’ ’0’ ’1’ ’Z’ ’H’ ’L’ ’W’ ’-’ 1'b0 1'b1</td>
</tr>
</tbody>
</table>
Array constants, expressed in any of the following formats

<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>VHDL # notation</td>
<td>&lt;int&gt;#&lt;alphanum&gt;[#]</td>
</tr>
<tr>
<td>Example:</td>
<td>16#abc123#</td>
</tr>
<tr>
<td>VHDL bitstring</td>
<td>&quot;(U</td>
</tr>
<tr>
<td>Example:</td>
<td>&quot;11010X11&quot;</td>
</tr>
<tr>
<td>VLOG notation</td>
<td>[-][&lt;int&gt;]'(b</td>
</tr>
<tr>
<td>(where &lt;alphanum&gt; includes 0-9, a-f, A-F and '-')</td>
<td>Example: 12’hc91 (This is the preferred notation because it removes the ambiguity about the number of bits.)</td>
</tr>
<tr>
<td>Based notation</td>
<td>0x..., 0X..., 0o..., 0O..., 0b..., OB...</td>
</tr>
<tr>
<td>ModelSim automatically zero fills unspecified upper bits.</td>
<td></td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of a signal</td>
<td>The name may be a simple name, a VHDL or VLOG style extended identifier, or a VHDL or VLOG style path. The signal must be one of the following types: -- VHDL signal of type INTEGER, REAL or TIME -- VHDL signal of type std_logic or bit -- VHDL signal of type user-defined enumeration -- VLOG net, VLOG register, VLOG integer, or VLOG real</td>
</tr>
<tr>
<td>NOW</td>
<td>Returns the value of time at the current location in the WLF file as the WLF file is being scanned (not the most recent simulation time).</td>
</tr>
</tbody>
</table>

Array variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of a signal</td>
<td>-- VHDL signals of type bit_vector or std_logic_vector -- VLOG register -- VLOG net array</td>
</tr>
<tr>
<td></td>
<td>A subrange or index may be specified in either VHDL or VLOG syntax. Examples: mysignal(1 to 5), mysignal[1:5], mysignal (4), mysignal [4]</td>
</tr>
</tbody>
</table>
**Signal attributes**

- `<name>`’event
- `<name>`’rising
- `<name>`’falling
- `<name>`’delayed()
- `<name>`’hasX

The `’delayed` attribute lets you assign a delay to a VHDL signal. To assign a delay to a signal in Verilog, use "#" notation in a subexpression (e.g., `#-10 /top/signalA`).

The `hasX` attribute lets you search for signals, nets, or registers that contains an X (unknown) value.

See "Examples" (CR-25) below for further details on `’delayed` and `’hasX`.

**Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;&amp;</code></td>
<td>boolean and</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>!</code></td>
<td>boolean not</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal</td>
</tr>
<tr>
<td><code>===</code></td>
<td>exact equal</td>
</tr>
<tr>
<td><code>!==</code></td>
<td>exact not equal</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal</td>
</tr>
<tr>
<td><code>not/NOT</code></td>
<td>unary bitwise inversion</td>
</tr>
<tr>
<td><code>and/AND/&amp;#</code></td>
<td>bitwise and</td>
</tr>
<tr>
<td><code>nand/NAND</code></td>
<td>bitwise nand</td>
</tr>
<tr>
<td>`or/OR/</td>
<td>`</td>
</tr>
<tr>
<td><code>nor/NOR</code></td>
<td>bitwise nor</td>
</tr>
<tr>
<td><code>xor/XOR</code></td>
<td>bitwise xor</td>
</tr>
<tr>
<td><code>xnor/XNOR</code></td>
<td>bitwise xnor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sll/SLL</code></td>
<td>shift left logical</td>
</tr>
<tr>
<td><code>sla/SLA</code></td>
<td>shift left arithmetic</td>
</tr>
<tr>
<td><code>srl/SRL</code></td>
<td>shift right logical</td>
</tr>
<tr>
<td><code>sra/SRA</code></td>
<td>shift right arithmetic</td>
</tr>
<tr>
<td><code>ror/ROR</code></td>
<td>rotate right</td>
</tr>
<tr>
<td><code>rol/ROL</code></td>
<td>rotate left</td>
</tr>
<tr>
<td><code>+</code></td>
<td>arithmetic add</td>
</tr>
<tr>
<td><code>-</code></td>
<td>arithmetic subtract</td>
</tr>
<tr>
<td><code>*</code></td>
<td>arithmetic multiply</td>
</tr>
<tr>
<td><code>/</code></td>
<td>arithmetic divide</td>
</tr>
<tr>
<td><code>mod/MOD</code></td>
<td>arithmetic modulus</td>
</tr>
<tr>
<td><code>rem/REM</code></td>
<td>arithmetic remainder</td>
</tr>
<tr>
<td>`</td>
<td>&lt;vector_expr&gt;`</td>
</tr>
<tr>
<td><code>^&lt;vector_expr&gt;</code></td>
<td>XOR reduction</td>
</tr>
</tbody>
</table>
Note: Arithmetic operators use the std_logic_arith package.

Casting

<table>
<thead>
<tr>
<th>Casting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(bool)</td>
<td>convert to boolean</td>
</tr>
<tr>
<td>(boolean)</td>
<td>convert to boolean</td>
</tr>
<tr>
<td>(int)</td>
<td>convert to integer</td>
</tr>
<tr>
<td>(integer)</td>
<td>convert to integer</td>
</tr>
<tr>
<td>(real)</td>
<td>convert to real</td>
</tr>
<tr>
<td>(time)</td>
<td>convert to 64-bit integer</td>
</tr>
<tr>
<td>(std_logic)</td>
<td>convert to 9-state signal value</td>
</tr>
<tr>
<td>(signed)</td>
<td>convert to signed vector</td>
</tr>
<tr>
<td>(unsigned)</td>
<td>convert to unsigned vector</td>
</tr>
<tr>
<td>(std_logic_vector)</td>
<td>convert to unsigned vector</td>
</tr>
</tbody>
</table>

Examples

/\top/bus & $bit_mask
This expression takes the bitwise AND function of signal /\top/bus and the array constant contained in the global Tcl variable bit_mask.

clk'event && (/\top/xyz == 16'hffae)
This expression evaluates to a boolean true when signal clk changes and signal /\top/xyz is equal to hex ffae; otherwise is false.

clk'rising && (mystate == reading) && (/\top/u3/addr == 32'habcd1234)
Evaluates to a boolean true when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /\top/u3/addr is equal to the specified 32-bit hex constant; otherwise is false.

(/\top/u3/addr and 32'hff000000) == 32'hac000000
Evaluates to a boolean true when the upper 8 bits of the 32-bit signal /\top/u3/adder equals hex ac.

/\top/signalA'delayed(10ns)
This expression returns /\top/signalA delayed by 10 ns.

/\top/signalA'delayed(10 ns) && /\top/signalB
This expression takes the logical AND of a delayed /\top/signalA with the undelayed /\top/signalB.
virtual function { (#-10 /top/signalA) && /top/signalB} 
mySignalB_AND_DelayedSignalA

This evaluates /top/signalA at 10 simulation time steps before the current time, and takes 
the logical AND of the result with the current value of /top/signalB. The '#' notation uses 
positive numbers for looking into the future, and negative numbers for delay. This 
notation does not support the use of time units.

((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing) 
Evaluates to a boolean true when WLF file time is between 23 and 54 microseconds, clk 
just changed from low to high, and signal mode is enumeration writing.

searchlog -expr {dbus'hasX} {0 ns} dbus 
Searches for an 'X' in dbus. This is equivalent to the expression: 
{dbus(0) == 'x'} || 
{dbus(1) == 'x'} . . . . This makes it possible to search for X values without having to write 
a type specific literal.
The commands here are entered either in macro files or on the command line of the Main window. Some commands are automatically entered on the command line when you use the ModelSim graphical user interface.

Note that in addition to the simulation commands documented in this section, you can use the Tcl commands described in the Tcl man pages (use the Main window menu selection: Help > Tcl Man Pages).

▶ **Note:** ModelSim commands are case sensitive. Type them as they are shown in this reference.
# Command reference table

The following table provides a brief description of each ModelSim command. Command details, arguments and examples can be found at the page numbers given in the Command name column.

<table>
<thead>
<tr>
<th>Command name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.main clear (CR-37)</td>
<td>clears the Main window transcript</td>
</tr>
<tr>
<td>.wave.tree interrupt (CR-38)</td>
<td>halts the drawing of waves in the Wave window</td>
</tr>
<tr>
<td>.wave.tree zoomfull (CR-39)</td>
<td>zooms the Wave window from time zero to the current simulation time</td>
</tr>
<tr>
<td>.wave.tree zoomin (CR-40)</td>
<td>zooms in the Wave window by the specified factor</td>
</tr>
<tr>
<td>.wave.tree zoomlast (CR-41)</td>
<td>zooms the Wave window to the setting of the last zoom change</td>
</tr>
<tr>
<td>.wave.tree zoomout (CR-42)</td>
<td>zooms out the Wave window by the specified factor</td>
</tr>
<tr>
<td>.wave.tree zoomrange (CR-43)</td>
<td>zooms the Wave to the specified range</td>
</tr>
<tr>
<td>abort (CR-44)</td>
<td>halts the execution of a macro file interrupted by a breakpoint or error</td>
</tr>
<tr>
<td>add button (CR-45)</td>
<td>adds a user-defined button to the Main window button bar</td>
</tr>
<tr>
<td>add dataflow (CR-47)</td>
<td>adds the specified item to the Dataflow window</td>
</tr>
<tr>
<td>add list (CR-48)</td>
<td>lists VHDL signals and variables, and Verilog nets and registers, and their values in the List window</td>
</tr>
<tr>
<td>add log</td>
<td>also known as the log command; see log (CR-166)</td>
</tr>
<tr>
<td>add_menu (CR-51)</td>
<td>adds a menu to the menu bar of the specified window, using the specified menu name</td>
</tr>
<tr>
<td>add_menucb (CR-53)</td>
<td>creates a checkbox within the specified menu of the specified window</td>
</tr>
<tr>
<td>add_menuitem (CR-54)</td>
<td>creates a menu item within the specified menu of the specified window</td>
</tr>
<tr>
<td>add_separator (CR-55)</td>
<td>adds a separator as the next item in the specified menu path in the specified window</td>
</tr>
<tr>
<td>add_submenu (CR-56)</td>
<td>creates a cascading submenu within the specified menu_path of the specified window</td>
</tr>
<tr>
<td>add wave (CR-57)</td>
<td>adds VHDL signals and variables, and Verilog nets and registers to the Wave window</td>
</tr>
<tr>
<td>alias (CR-61)</td>
<td>creates a new Tcl procedure that evaluates the specified commands</td>
</tr>
<tr>
<td>batch_mode (CR-62)</td>
<td>returns a 1 if ModelSim is operating in batch mode, otherwise returns a 0</td>
</tr>
<tr>
<td>bd (CR-63)</td>
<td>deletes a breakpoint</td>
</tr>
<tr>
<td>bookmark add wave (CR-64)</td>
<td>adds a bookmark to the specified Wave window</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bookmark delete wave (CR-65)</td>
<td>deletes bookmarks from the specified Wave window</td>
</tr>
<tr>
<td>bookmark goto wave (CR-66)</td>
<td>zooms and scrolls a Wave window using the specified bookmark</td>
</tr>
<tr>
<td>bookmark list wave (CR-67)</td>
<td>displays a list of available bookmarks</td>
</tr>
<tr>
<td>bp (CR-68)</td>
<td>sets a breakpoint</td>
</tr>
<tr>
<td>cd (CR-71)</td>
<td>changes the ModelSim local directory to the specified directory</td>
</tr>
<tr>
<td>change (CR-72)</td>
<td>modifies the value of a VHDL variable or Verilog register variable</td>
</tr>
<tr>
<td>change_menu_cmd (CR-73)</td>
<td>changes the command to be executed for a specified menu item label, in the specified menu, in the specified window</td>
</tr>
<tr>
<td>check contention add (CR-74)</td>
<td>enables contention checking for the specified nodes</td>
</tr>
<tr>
<td>check contention config (CR-75)</td>
<td>writes checking messages to a file</td>
</tr>
<tr>
<td>check contention off (CR-76)</td>
<td>disables contention checking for the specified nodes</td>
</tr>
<tr>
<td>check float add (CR-77)</td>
<td>enables float checking for the specified nodes</td>
</tr>
<tr>
<td>check float config (CR-78)</td>
<td>writes checking messages to a file</td>
</tr>
<tr>
<td>check float off (CR-79)</td>
<td>disables float checking for the specified nodes</td>
</tr>
<tr>
<td>check stable off (CR-80)</td>
<td>disables stability checking</td>
</tr>
<tr>
<td>check stable on (CR-81)</td>
<td>enables stability checking on the entire design</td>
</tr>
<tr>
<td>checkpoint (CR-82)</td>
<td>saves the state of your simulation</td>
</tr>
<tr>
<td>compare add (CR-83)</td>
<td>compares signals in a reference design against signals in a test design</td>
</tr>
<tr>
<td>compare annotate (CR-86)</td>
<td>marks a compare difference as &quot;ignore&quot; or tags it with a text message</td>
</tr>
<tr>
<td>compare clock (CR-87)</td>
<td>defines a clock to be used with clocked-mode comparisons</td>
</tr>
<tr>
<td>compare configure (CR-89)</td>
<td>modifies options for compare signals or regions</td>
</tr>
<tr>
<td>compare continue (CR-90)</td>
<td>continues difference computation that had been suspended</td>
</tr>
<tr>
<td>compare delete (CR-91)</td>
<td>deletes a signal or region from the current comparison</td>
</tr>
<tr>
<td>compare end (CR-92)</td>
<td>closes the currently open comparison</td>
</tr>
<tr>
<td>compare info (CR-93)</td>
<td>lists the results of the comparison</td>
</tr>
<tr>
<td>compare list (CR-95)</td>
<td>lists all the compare add commands currently in effect</td>
</tr>
<tr>
<td>compare options (CR-96)</td>
<td>sets defaults for options used in other compare commands</td>
</tr>
<tr>
<td>compare reload (CR-99)</td>
<td>reloads a comparison previously saved with the compare savediffs command</td>
</tr>
<tr>
<td>compare reset (CR-100)</td>
<td>clears the current compare differences</td>
</tr>
<tr>
<td>compare run (CR-101)</td>
<td>runs the comparison on selected signals</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>compare savediffs (CR-102)</td>
<td>saves comparison differences to a file that can be reloaded later</td>
</tr>
<tr>
<td>compare saverules (CR-103)</td>
<td>saves comparison setup information to a file that can be reloaded later</td>
</tr>
<tr>
<td>compare see (CR-104)</td>
<td>displays a comparison difference in the Wave window</td>
</tr>
<tr>
<td>compare start (CR-106)</td>
<td>starts a new dataset comparison</td>
</tr>
<tr>
<td>compare stop (CR-108)</td>
<td>halts active difference computation</td>
</tr>
<tr>
<td>compare update (CR-109)</td>
<td>updates the comparison differences</td>
</tr>
<tr>
<td>configure (CR-110)</td>
<td>invokes the List or Wave widget configure command for the current default List or Wave window</td>
</tr>
<tr>
<td>context (CR-114)</td>
<td>provides several operations on a context’s name</td>
</tr>
<tr>
<td>coverage clear (CR-116)</td>
<td>clears all coverage data obtained during previous run commands</td>
</tr>
<tr>
<td>coverage reload (CR-121)</td>
<td>seeds the coverage statistics with the output of a previous coverage report command</td>
</tr>
<tr>
<td>coverage exclude clear (CR-117)</td>
<td>unloads the current exclusion filter file</td>
</tr>
<tr>
<td>coverage exclude disable (CR-118)</td>
<td>disables the current exclusion filter file</td>
</tr>
<tr>
<td>coverage exclude enable (CR-119)</td>
<td>enables a previously disabled exclusion filter file</td>
</tr>
<tr>
<td>coverage exclude load (CR-120)</td>
<td>loads an exclusion filter file</td>
</tr>
<tr>
<td>coverage report (CR-122)</td>
<td>produces a textual output of the coverage statistics that have been gathered up to this point</td>
</tr>
<tr>
<td>dataset alias (CR-123)</td>
<td>assigns an additional name to a dataset</td>
</tr>
<tr>
<td>dataset clear (CR-124)</td>
<td>clears the current simulation WLF file</td>
</tr>
<tr>
<td>dataset close (CR-125)</td>
<td>closes a dataset</td>
</tr>
<tr>
<td>dataset info (CR-126)</td>
<td>reports information about the specified dataset</td>
</tr>
<tr>
<td>dataset list (CR-127)</td>
<td>lists the open dataset(s)</td>
</tr>
<tr>
<td>dataset open (CR-128)</td>
<td>opens a dataset and references it by a logical name</td>
</tr>
<tr>
<td>dataset rename (CR-129)</td>
<td>changes the logical name of an opened dataset</td>
</tr>
<tr>
<td>dataset save (CR-130)</td>
<td>saves data from the current WLF file to a specified file</td>
</tr>
<tr>
<td>dataset snapshot (CR-131)</td>
<td>saves data from the current WLF file at a specified interval</td>
</tr>
<tr>
<td>delete (CR-133)</td>
<td>removes HDL items from either the List or Wave window</td>
</tr>
<tr>
<td>describe (CR-134)</td>
<td>displays information about the specified HDL item</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>disablebp (CR-135)</td>
<td>turns off breakpoints and when commands are turned off by the disablebp command (CR-135)</td>
</tr>
<tr>
<td>disable_menu (CR-136)</td>
<td>disables the specified menu within the specified window</td>
</tr>
<tr>
<td>disable_menuitem (CR-137)</td>
<td>disables the specified menu item within the specified menu_path of the specified window</td>
</tr>
<tr>
<td>do (CR-138)</td>
<td>executes commands contained in a macro file</td>
</tr>
<tr>
<td>down (CR-139)</td>
<td>searches for signal transitions or values in the specified List window</td>
</tr>
<tr>
<td>drivers (CR-141)</td>
<td>displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net</td>
</tr>
<tr>
<td>dumplog64 (CR-142)</td>
<td>dumps the contents of the vsim.wlf file in a readable format</td>
</tr>
<tr>
<td>echo (CR-143)</td>
<td>displays a specified message in the Main window</td>
</tr>
<tr>
<td>edit (CR-144)</td>
<td>invokes the editor specified by the EDITOR environment variable</td>
</tr>
<tr>
<td>enablebp (CR-145)</td>
<td>turns on breakpoints and when commands turned off by the disablebp command (CR-135)</td>
</tr>
<tr>
<td>enable_menu (CR-146)</td>
<td>enables a previously-disabled menu</td>
</tr>
<tr>
<td>enable_menuitem (CR-147)</td>
<td>enables a previously-disabled menu item</td>
</tr>
<tr>
<td>environment (CR-148)</td>
<td>displays or changes the current dataset and region environment</td>
</tr>
<tr>
<td>examine (CR-149)</td>
<td>examines one or more HDL items, and displays current values (or the values at a specified previous time) in the Main window</td>
</tr>
<tr>
<td>exit (CR-152)</td>
<td>exits the simulator and the ModelSim application</td>
</tr>
<tr>
<td>find (CR-153)</td>
<td>displays the full pathnames of all HDL items in the design whose names match the name specification you provide</td>
</tr>
<tr>
<td>force (CR-156)</td>
<td>allows you to apply stimulus to VHDL signals and Verilog nets and registers, interactively</td>
</tr>
<tr>
<td>getactivecursortime (CR-159)</td>
<td>gets the time of the active cursor in the Wave window</td>
</tr>
<tr>
<td>getactivemarkertime (CR-160)</td>
<td>gets the time of the active marker in the List window</td>
</tr>
<tr>
<td>help (CR-161)</td>
<td>displays in the Main window a brief description and syntax for the specified command</td>
</tr>
<tr>
<td>history (CR-162)</td>
<td>lists the commands executed during the current session</td>
</tr>
<tr>
<td>lecho (CR-163)</td>
<td>takes one or more Tcl lists as arguments and pretty-prints them to the Main window</td>
</tr>
<tr>
<td>left (CR-164)</td>
<td>searches left (previous) for signal transitions or values in the specified Wave window</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>log (CR-166)</td>
<td>creates a wave log format (WLF) file containing simulation data for all HDL items whose names match the provided specifications</td>
</tr>
<tr>
<td>lshift (CR-168)</td>
<td>takes a Tcl list as argument and shifts it in-place one place to the left, eliminating the 0th element</td>
</tr>
<tr>
<td>lsublist (CR-169)</td>
<td>returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern</td>
</tr>
<tr>
<td>macro_option (CR-170)</td>
<td>controls the speed and delay of macro (DO file) playback, plus the level of debugging feedback</td>
</tr>
<tr>
<td>modelsim (CR-171)</td>
<td>starts the ModelSim GUI without prompting you to load a design; valid only for Windows platforms</td>
</tr>
<tr>
<td>next (CR-172)</td>
<td>continues a search; see the search command (CR-212)</td>
</tr>
<tr>
<td>noforce (CR-173)</td>
<td>removes the effect of any active force (CR-156) commands on the selected HDL items</td>
</tr>
<tr>
<td>nolog (CR-174)</td>
<td>suspends writing of data to the WLF file for the specified signals</td>
</tr>
<tr>
<td>notepad (CR-176)</td>
<td>opens a simple text editor</td>
</tr>
<tr>
<td>noview (CR-177)</td>
<td>closes a window in the ModelSim GUI</td>
</tr>
<tr>
<td>nowhen (CR-178)</td>
<td>deactivates selected when (CR-314) commands</td>
</tr>
<tr>
<td>onbreak (CR-179)</td>
<td>specifies command(s) to be executed when running a macro that encounters a breakpoint in the source code</td>
</tr>
<tr>
<td>onElabError (CR-180)</td>
<td>specifies one or more commands to be executed when an error is encountered during elaboration</td>
</tr>
<tr>
<td>onerror (CR-181)</td>
<td>specifies one or more commands to be executed when a running macro encounters an error</td>
</tr>
<tr>
<td>pause (CR-182)</td>
<td>interrupts the execution of a macro</td>
</tr>
<tr>
<td>play (CR-183)</td>
<td>plays a sequence of keyboard and mouse actions, which were previously saved to a file with the record command (CR-201)</td>
</tr>
<tr>
<td>power add (CR-184)</td>
<td>specifies the signals or nets to track for power information</td>
</tr>
<tr>
<td>power report (CR-185)</td>
<td>writes out the power information for the specified signals or nets</td>
</tr>
<tr>
<td>power reset (CR-186)</td>
<td>resets power information to zero for the signals or nets specified with the power add command (CR-184)</td>
</tr>
<tr>
<td>printenv (CR-187)</td>
<td>echoes to the Main window the current names and values of all environment variables</td>
</tr>
<tr>
<td>profile clear (CR-188)</td>
<td>clears any data that has been gathered during previous run commands</td>
</tr>
</tbody>
</table>
### Command reference table

<table>
<thead>
<tr>
<th>Command name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile interval</td>
<td>selects the frequency with which the profiler collects samples during a run command</td>
</tr>
<tr>
<td>profile off</td>
<td>discontinues runtime profiling</td>
</tr>
<tr>
<td>profile on</td>
<td>enables runtime analysis of where your simulation is spending its time</td>
</tr>
<tr>
<td>profile option</td>
<td>allows various profiling options to be changed</td>
</tr>
<tr>
<td>profile report</td>
<td>produces a textual output of the profiling statistics that have been gathered up to this point</td>
</tr>
<tr>
<td>project</td>
<td>performs common operations on new projects</td>
</tr>
<tr>
<td>property list</td>
<td>changes one or more properties of the specified signal, net or register in the List window (UM-204)</td>
</tr>
<tr>
<td>property wave</td>
<td>changes one or more properties of the specified signal, net or register in the Wave window (UM-246)</td>
</tr>
<tr>
<td>pwd</td>
<td>displays the current directory path in the Main window</td>
</tr>
<tr>
<td>quietly</td>
<td>turns off transcript echoing for the specified command</td>
</tr>
<tr>
<td>quit</td>
<td>exits the simulator</td>
</tr>
<tr>
<td>radix</td>
<td>specifies the default radix to be used</td>
</tr>
<tr>
<td>record</td>
<td>starts recording a replayable trace of all keyboard and mouse actions</td>
</tr>
<tr>
<td>report</td>
<td>displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation</td>
</tr>
<tr>
<td>restart</td>
<td>reloads the design elements and resets the simulation time to zero</td>
</tr>
<tr>
<td>restore</td>
<td>restores the state of a simulation that was saved with a checkpoint command (CR-82) during the current invocation of vsim</td>
</tr>
<tr>
<td>resume</td>
<td>resumes execution of a macro file after a pause command (CR-182), or a breakpoint</td>
</tr>
<tr>
<td>right</td>
<td>searches right (next) for signal transitions or values in the specified Wave window</td>
</tr>
<tr>
<td>run</td>
<td>advances the simulation by the specified number of timesteps</td>
</tr>
<tr>
<td>search</td>
<td>searches the specified window for one or more items matching the specified pattern(s)</td>
</tr>
<tr>
<td>searchlog</td>
<td>searches one or more of the currently open logfiles for a specified condition</td>
</tr>
<tr>
<td>seetime</td>
<td>scrolls the List or Wave window to make the specified time visible</td>
</tr>
<tr>
<td>shift</td>
<td>shifts macro parameter values down one place</td>
</tr>
<tr>
<td>show</td>
<td>lists HDL items and subregions visible from the current environment</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
</tr>
<tr>
<td>simstats (CR-219)</td>
<td>reports performance-related statistics about active simulations</td>
</tr>
<tr>
<td>splitio (CR-220)</td>
<td>operates on a VHDL inout or out port to create a new signal having the same name as the port suffixed with &quot;_o&quot;</td>
</tr>
<tr>
<td>status (CR-221)</td>
<td>lists all currently interrupted macros</td>
</tr>
<tr>
<td>step (CR-222)</td>
<td>steps to the next HDL statement</td>
</tr>
<tr>
<td>stop (CR-223)</td>
<td>stops simulation in batch files; used with the when command (CR-314)</td>
</tr>
<tr>
<td>tb (CR-224)</td>
<td>displays a stack trace for the current process in the Main window</td>
</tr>
<tr>
<td>toggle add (CR-225)</td>
<td>enables collection of toggle statistics for the specified nodes</td>
</tr>
<tr>
<td>toggle report (CR-226)</td>
<td>displays to the screen a list of all nodes that have not transitioned to both 0 and 1 at least once</td>
</tr>
<tr>
<td>toggle reset (CR-227)</td>
<td>resets the toggle counts to zero for the specified nodes</td>
</tr>
<tr>
<td>transcribe (CR-228)</td>
<td>displays a command in the Main window, then executes the command</td>
</tr>
<tr>
<td>transcript (CR-229)</td>
<td>controls echoing of commands executed in a macro file; also works at top level in batch mode</td>
</tr>
<tr>
<td>tssi2mti (CR-230)</td>
<td>converts a vector file in Fluence Technology (formerly TSSI) Standard Events Format into a sequence of force (CR-156) and run (CR-210) commands</td>
</tr>
<tr>
<td>up (CR-231)</td>
<td>searches for signal transitions or values in the specified List window</td>
</tr>
<tr>
<td>vcd add (CR-233)</td>
<td>adds the specified items to the VCD file</td>
</tr>
<tr>
<td>vcd checkpoint (CR-234)</td>
<td>dumps the current values of all VCD variables to the VCD file</td>
</tr>
<tr>
<td>vcd comment (CR-235)</td>
<td>inserts the specified comment in the VCD file</td>
</tr>
<tr>
<td>vcd dumpports (CR-236)</td>
<td>creates a VCD file that captures port driver data</td>
</tr>
<tr>
<td>vcd dumpportsall (CR-238)</td>
<td>creates a checkpoint in the VCD file that shows the current value of all selected ports</td>
</tr>
<tr>
<td>vcd dumpportsflush (CR-239)</td>
<td>flushes the VCD buffer to the VCD file</td>
</tr>
<tr>
<td>vcd dumpportslimit (CR-240)</td>
<td>specifies the maximum size of the VCD file</td>
</tr>
<tr>
<td>vcd dumpportsoff (CR-241)</td>
<td>turns off VCD dumping and records all dumped port values as x</td>
</tr>
<tr>
<td>vcd dumpportson (CR-242)</td>
<td>turns on VCD dumping and records the current value of all selected ports</td>
</tr>
<tr>
<td>vcd file (CR-243)</td>
<td>specifies the filename and state mapping for the VCD file created by a vcd add command (CR-233)</td>
</tr>
<tr>
<td>vcd files (CR-245)</td>
<td>specifies the filename and state mapping for the VCD file created by a vcd add command (CR-233); supports multiple VCD files</td>
</tr>
<tr>
<td>vcd flush (CR-247)</td>
<td>flushes the contents of the VCD file buffer to the VCD file</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>vcd limit (CR-248)</td>
<td>specifies the maximum size of the VCD file</td>
</tr>
<tr>
<td>vcd off (CR-249)</td>
<td>turns off VCD dumping and records all VCD variable values as x</td>
</tr>
<tr>
<td>vcd on (CR-250)</td>
<td>turns on VCD dumping and records the current values of all VCD variables</td>
</tr>
<tr>
<td>vcd2wlf (CR-251)</td>
<td>translates VCD files into WLF files</td>
</tr>
<tr>
<td>vcom (CR-252)</td>
<td>compiles VHDL design units</td>
</tr>
<tr>
<td>vdel (CR-258)</td>
<td>deletes a design unit from a specified library</td>
</tr>
<tr>
<td>vdir (CR-259)</td>
<td>lists the contents of a design library</td>
</tr>
<tr>
<td>verror (CR-260)</td>
<td>prints a detailed description of a message number</td>
</tr>
<tr>
<td>vgencomp (CR-261)</td>
<td>writes a Verilog module’s equivalent VHDL component declaration to standard output</td>
</tr>
<tr>
<td>view (CR-263)</td>
<td>opens a ModelSim window and brings it to the front of the display</td>
</tr>
<tr>
<td>virtual count (CR-265)</td>
<td>counts the number of currently defined virtuals that were not read in using a macro file</td>
</tr>
<tr>
<td>virtual define (CR-266)</td>
<td>prints the definition of the virtual signal or function in the form of a command that can be used to re-create the object</td>
</tr>
<tr>
<td>virtual delete (CR-267)</td>
<td>removes the matching virtuals</td>
</tr>
<tr>
<td>virtual describe (CR-268)</td>
<td>prints a complete description of the data type of one or more virtual signals</td>
</tr>
<tr>
<td>virtual expand (CR-269)</td>
<td>produces a list of all the non-virtual objects contained in the virtual signal(s)</td>
</tr>
<tr>
<td>virtual function (CR-270)</td>
<td>creates a new signal that consists of logical operations on existing signals and simulation time</td>
</tr>
<tr>
<td>virtual hide (CR-273)</td>
<td>sets a flag in the specified real or virtual signals so that the signals do not appear in the Signals window</td>
</tr>
<tr>
<td>virtual log (CR-274)</td>
<td>causes the sim-mode dependent signals of the specified virtual signals to be logged by the simulator</td>
</tr>
<tr>
<td>virtual nohide (CR-276)</td>
<td>resets the flag set by a virtual hide command</td>
</tr>
<tr>
<td>virtual nolog (CR-277)</td>
<td>stops the logging of the specified virtual signals</td>
</tr>
<tr>
<td>virtual region (CR-279)</td>
<td>creates a new user-defined design hierarchy region</td>
</tr>
<tr>
<td>virtual save (CR-280)</td>
<td>saves the definitions of virtuals to a file</td>
</tr>
<tr>
<td>virtual show (CR-281)</td>
<td>lists the full path names of all the virtuals explicitly defined</td>
</tr>
<tr>
<td>virtual signal (CR-282)</td>
<td>creates a new signal that consists of concatenations of signals and subelements</td>
</tr>
<tr>
<td>virtual type (CR-285)</td>
<td>creates a new enumerated type</td>
</tr>
<tr>
<td>Command name</td>
<td>Action</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>vlib</code> (CR-287)</td>
<td>creates a design library</td>
</tr>
<tr>
<td><code>vlog</code> (CR-288)</td>
<td>compiles Verilog design units</td>
</tr>
<tr>
<td><code>vmake</code> (CR-296)</td>
<td>creates a makefile that can be used to reconstruct the specified library</td>
</tr>
<tr>
<td><code>vmap</code> (CR-297)</td>
<td>defines a mapping between a logical library name and a directory by modifying the <em>modelsim.ini</em> file</td>
</tr>
<tr>
<td><code>vsim</code> (CR-298)</td>
<td>loads a new design into the simulator</td>
</tr>
<tr>
<td><code>vsim&lt;info&gt;</code> (CR-312)</td>
<td>returns information about the current vsim executable</td>
</tr>
<tr>
<td><code>vsource</code> (CR-313)</td>
<td>specifies an alternative file to use for the current source file</td>
</tr>
<tr>
<td><code>when</code> (CR-314)</td>
<td>instructs ModelSim to perform actions when the specified conditions are met</td>
</tr>
<tr>
<td><code>where</code> (CR-318)</td>
<td>displays information about the system environment</td>
</tr>
<tr>
<td><code>wlf2log</code> (CR-319)</td>
<td>translates a ModelSim WLF file(<em>vsim.wlf</em>) to a QuickSim II logfile</td>
</tr>
<tr>
<td><code>wlfrecover</code> (CR-321)</td>
<td>attempts to repair incomplete WLF files</td>
</tr>
<tr>
<td><code>write cell_report</code> (CR-322)</td>
<td>creates a report of cell instances in the design that are optimized (-fast)</td>
</tr>
<tr>
<td><code>write format</code> (CR-323)</td>
<td>records the names and display options in a file of the HDL items currently being displayed in the List or Wave window</td>
</tr>
<tr>
<td><code>write list</code> (CR-325)</td>
<td>records the contents of the most recently opened or specified List window in a list output file</td>
</tr>
<tr>
<td><code>write preferences</code> (CR-326)</td>
<td>saves the current GUI preference settings to a Tcl preference file</td>
</tr>
<tr>
<td><code>write report</code> (CR-327)</td>
<td>prints a summary of the design being simulated</td>
</tr>
<tr>
<td><code>write transcript</code> (CR-328)</td>
<td>writes the contents of the Main window transcript to the specified file</td>
</tr>
<tr>
<td><code>write tssi</code> (CR-329)</td>
<td>records the contents of the default or specified List window in a “TSSI format” file</td>
</tr>
<tr>
<td><code>write wave</code> (CR-331)</td>
<td>records the contents of the most currently opened or specified Wave window in PostScript format</td>
</tr>
</tbody>
</table>
The `.main clear` command clears the transcript. The behavior is the same as the Main window File > Transcript > Clear Transcript menu selection.

Syntax

```
.main clear
```

Arguments

None.

See also

Main window (UM-173)
.wave.tree interrupt

The `.wave.tree interrupt` command halts the drawing of waves in the Wave window. This command can be useful when you have a large WLF file that is taking a long time to display.

Syntax

`.wave.tree interrupt`

Arguments

None.
The `.wave.tree zoomfull` command redraws the Wave window to show the entire simulation from time 0 to the current simulation time. The behavior is the same as the Wave window (UM-246) View > Zoom > Zoom Full menu selection.

Returns the zoom range as two time values.

**Syntax**

`.wave.tree zoomfull`

**Arguments**

None.

**See also**

`.wave.tree zoomin` (CR-40), `.wave.tree zoomlast` (CR-41), `.wave.tree zoomout` (CR-42), `.wave.tree zoomrange` (CR-43)

**Example**

```
.wave.tree zoomfull
# {0 ns}{2310 ns}
```
.wave.tree zoomin

The .wave.tree zoomin command allows you to zoom in the Wave window by some factor. The behavior is similar to the Wave window (UM-246) View > Zoom > Zoom In menu selection.

Returns the zoom range as two time values.

Syntax

```
.wave.tree zoomin
  <factor>
```

Arguments

`<factor>`
A number that specifies how much you want to zoom in the Wave window. Required.

See also

- .wave.tree zoomfull (CR-39)
- .wave.tree zoomlast (CR-41)
- .wave.tree zoomout (CR-42)
- .wave.tree zoomrange (CR-43)

Example

```
.wave.tree zoomin 2
# [577 ns][1733 ns]
```
.wave.tree zoomlast

The .wave.tree zoomlast command zooms the Wave window to the setting prior to the most recent zoom change. The behavior is the same as the Wave window (UM-246) View > Zoom > Zoom Last menu selection.

Returns the zoom range as two time values.

Syntax

.wave.tree zoomlast

Arguments

None.

See also

.wave.tree zoomfull (CR-39), .wave.tree zoomin (CR-40), .wave.tree zoomout (CR-42), .wave.tree zoomrange (CR-43)

Example

.wave.tree zoomlast
# {0 ns}{2310 ns}
The `.wave.tree zoomout` command allows you to zoom out the Wave window by some factor. The behavior is similar to the Wave window (UM-246) View > Zoom > Zoom Out menu selection.

Returns the zoom range as two time values.

**Syntax**

```
.wave.tree zoomout

<factor>
```

**Arguments**

<factor>

A number that specifies how much you want to zoom out the Wave window. Required.

**See also**

`.wave.tree zoomfull` (CR-39), `.wave.tree zoomin` (CR-40), `.wave.tree zoomlast` (CR-41), `.wave.tree zoomrange` (CR-43)

**Example**

```
.wave.tree zoomout 2
# {865 ns}[1445 ns]
```
.wave.tree zoomrange

The .wave.tree zoomrange command lets you set the zoom range for the Wave window. The behavior is the same as the Wave window View > Zoom > Zoom Range menu selection.

Returns the zoom range as two time values.

Syntax

.wave.tree zoomrange

[<time1> [<time2>]]

Arguments

<time1>
<time2>

time1 and time2 are floating point numbers that specify a zoom range. If neither number is specified, the command returns the current zoom range. If only time1 is specified, then the zoom range is set to start at 0 and end at time1.

Either range number may include an optional VHDL resolution time-unit. The resolution and range number must be enclosed in either quotes or curly brackets (see the example below). If not specified the resolution defaults to the UserTimeUnit (UM-450) set in the modelsim.ini file.

Examples

.wave.tree zoomrange {.5 us} {1.75 us}
# (500 ns) (1750 ns)

Zooms the Wave window between .5 us and 1.75 us and returns the zoom range in current simulator time units.

See also

.wave.tree zoomfull (CR-39), .wave.tree zoomin (CR-40), .wave.tree zoomlast (CR-41), .wave.tree zoomout (CR-42)
abort

The `abort` command halts the execution of a macro file interrupted by a breakpoint or error. When macros are nested, you may choose to abort the last macro only, abort a specified number of nesting levels, or abort all macros. The `abort` command may be used within a macro to return early.

Syntax

```
abort
   [ <n> | all ]
```

Arguments

```
<n> | all
```

An integer giving the number of nested macro levels to abort; `all` aborts all levels. Optional. Default is 1.

See also

`onbreak` (CR-179), `onElabError` (CR-180), `onerror` (CR-181)
add button

The `add button` command adds a user-defined button to the Main window button bar. New buttons are added to the right end of the bar. You can also add buttons with a ModelSim tool: "The Button Adder" (UM-310).

Returns the path name of the button widget created.

Syntax

```
add button <Text> <Cmd> [Disable | NoDisable] {{<option> <value> ...}}
```

Arguments

- `<Text>`
  The label to appear on the face of the button. Required.

- `<Cmd>`
  The command to be executed when the button is clicked with the left mouse button. To echo the command and display the return value in the Main window, prefix the command with the `transcribe` command (CR-228). `Transcribe` will also echo the results to the transcript window. Required.

- Disable | NoDisable
  If Disable, the button will be grayed-out during a run and not active. If NoDisable, the button will continue to be active during a run. Optional. The default is Disable.

- {{<option> <value> ...}}
  A list of option-value pairs that will be applied to the button widget. Optional. Any properties belonging to Tk button widgets may be set. Useful options are foreground color (`-fg`), background color (`-bg`), width (`-width`) and relief (`-relief`).

  For a complete list of available options, use the configure command addressed to the newly-created widget. For example:

  ```shell
  .controls.tb.button_7 config
  ```

**Note:** Because the arguments are positional, a Disable | NoDisable option must be specified in order to use the options argument.
Examples

add button pwd {transcribe pwd} NoDisable
Creates a button labeled "pwd" that invokes the transcribe command (CR-228) with the pwd Tcl command, and echoes the command and its results to the Main window (see graphic below). The button remains active during a run.

add button date {transcribe exec date} Disable {-fg blue -bg yellow -activebackground red}
Creates a button labeled "date" that echoes the system date to the Main window. The button is disabled during a run; its colors are: blue foreground, yellow background, and red active background.

add button doit {run 1000 ns; echo did it} Disable {-underline 1}
Creates a "doit" button and underlines the second character of the label, the "o" of "doit".

.controls.tb.button_7 config -command {run 10000} -bg red
Changes the button command to "run 10000" and changes the button background color to red.

See also

transcribe (CR-228), "The Button Adder" (UM-310) tool
add dataflow

The add dataflow command adds the specified process, signal, net, or register to the Dataflow window. Wildcards are allowed.

Syntax

add dataflow
   <item> [-window <wname>]

<item>
   Specifies a process, signal, net, or register that you want to add to the Dataflow window. Required. Multiple items separated by spaces may be specified. Wildcards are allowed. (Note that the WildcardFilter Tcl preference variable identifies types to ignore when matching items with wildcard patterns.)

-window <wname>
   Adds the items to the specified Dataflow window <wname> (e.g., dataflow2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the view command (CR-263) with the -new option to create a new window.

See also

Dataflow window (UM-186)
add list

The add list command lists VHDL signals and variables and Verilog nets and registers in the List window, along with their associated values. User-defined buses may also be added for either language.

If no port mode is specified, add list will display all items in the selected region with names matching the item name specification.

Limitations: VHDL variables and Verilog memories can be listed using the variable’s full name only (no wildcards).

Syntax

```
add list
   [-allowconstants] [-in] [-inout] [-internal]
   [[<item_name> | {<item_name> {sig1 sig2 sig3 ...}}] ...] ...
   [-recursive] [-width <n>] [-window <wname>]
```

Arguments

-allowconstants
  For use with wildcard searches. Specifies that constants matching the wildcard search should be added to the List window. Optional. By default, constants are ignored because they do not change.

-in
  For use with wildcard searches. Specifies that the scope of the search is to include ports of mode IN if they match the item name specification. Optional.

-inout
  For use with wildcard searches. Specifies that the scope of the search is to include ports of mode INOUT if they match the item name specification. Optional.

-internal
  For use with wildcard searches. Specifies that the scope of the search is to include internal items (non-port items) if they match the item name specification. VHDL variables are not selected. Optional.

=item_name
  Specifies the name of the item to be listed. Optional. Wildcard characters are allowed. (Note that the WildcardFilter Tcl preference variable identifies types to ignore when matching items with wildcard patterns.) Variables may be added if preceded by the process name. For example,
  add list myproc/int1

{<item_name> {sig1 sig2 sig3 ...}}
  Creates a user-defined bus in place of item name; ‘sig’ are signals to be concatenated within the user-defined bus. Optional. Specified items may be either scalars or various sized arrays as long as they have the same element enumeration type.
-label <name>
Specifies an alternative signal name to be displayed as a column heading in the listing. Optional. This alternative name is not valid in a force (CR-156) or examine (CR-149) command; however, it can be used in a search command (CR-212) with the list option.

-nodelta
Specifies that the delta column not be displayed when adding signals to the List window. Optional. Identical to configure list -delta none.

-notrigger
Specifies that items are to be listed, but does not cause the List window to be updated when the item changes. Optional.

-out
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode OUT if they match the item_name specification. Optional.

-ports
For use with wildcard searches. Specifies that the scope of the search is to include all ports. Optional. Has the same effect as specifying -in, -out, and -inout together.

-<radix>
Specifies the radix for the items that follow in the command. Optional. Valid entries (or unique abbreviations) are:

  binary
  octal
  decimal (or signed)
  unsigned
  hexadecimal
  ascii
  symbolic
  default

If no radix is specified for an enumerated type, the default representation is used. You can change the default radix for the current simulation using the radix command (CR-200). You can change the default radix permanently by editing the DefaultRadix (UM-447) variable in the modelsim.ini file.

If you specify a radix for an array of a VHDL enumerated type, ModelSim converts each signal value to 1, 0, Z, or X.

-recursive
For use with wildcard searches. Specifies that the scope of the search is to descend recursively into subregions. Optional; if omitted, the search is limited to the selected region.

-trigger
Specifies that items are to be listed and causes the List window to be updated when the items change. Optional. Default.

-width <n>
Specifies the column width in characters. Optional.

-window <wname>
Adds HDL items to the specified List window <wname> (e.g., list2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the view command (CR-263) with the -new option to create a new window.
Examples

add list -r */
   Lists all items in the design.

add list *
   Lists all items in the region.

add list -in *
   Lists all input ports in the region.

add list a -label sig /top/lower/sig {array_sig(9 to 23)}
   Displays a List window containing three columns headed $a$, $sig$, and $array_sig(9$ to $23)$.

add list clk -notrigger a b c d
   Lists $clk$, $a$, $b$, $c$, and $d$ only when $clk$ changes.

config list -strobeperiod {100 ns} -strobestart {0 ns} -usestrobe 1
add list -notrigger clk a b c d
   Lists $clk$, $a$, $b$, $c$, and $d$ every 100 ns.

add list -hex {mybus {msb {opcode(8 downto 1)} data}}
   Creates a user-defined bus named "mybus" consisting of three signals; the bus is displayed in hex.

add list vec1 -hex vec2 -dec vec3 vec4
   Lists the item $vec1$ using symbolic values, lists $vec2$ in hexadecimal, and lists $vec3$ and $vec4$ in decimal.

See also

add wave (CR-57), log (CR-166), "Extended identifiers" (CR-15)
add_menu

The `add_menu` command adds a menu to the menu bar of the specified window, using the specified menu name. Use the `add_menuitem` (CR-54), `add_separator` (CR-55), `add_menucb` (CR-53), and `add_submenu` (CR-56) commands to complete the menu.

Returns the full Tk pathname of the new menu.

Color and other Tk properties of the menu may be changed, after creating the menu, using the Tk menu widget configure command.

Syntax

```
add_menu
   <window_name>  <menu_name>  [ <shortcut>  [ -hide_menubutton ] ]
```

Arguments

- `<window_name>`
  Tk path of the window to contain the menu. Required.
  Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

- `<menu_name>`
  Name to be given to the Tk menu widget. Required.

- `<shortcut>`
  Number of the letter in the menu name that is to be used as the shortcut. Numbering starts with 0 (i.e., first letter = 0, second letter = 1, third letter = 2, etc.). Optional unless you specify `-hide_menubutton`, in which case `<shortcut>` is required. Default is "-1", which indicates no shortcut is to be used.

- `-hide_menubutton`
  Causes the new menu not to be displayed. Optional. You can add the menu later by calling `tk_popup` on the menu path widget. Note that you must specify `<shortcut>` if you specify `-hide_menubutton`.

Examples

The following Tcl code is an example of creating user-customized menus. It adds a menu containing a top-level item labeled "Do My Own Thing...", which prints "my_own_thing.signals", and adds a cascading submenu labeled "changeCase" with two entries, "To Upper" and "To Lower", which echo "my_to_upper" and "my_to_lower" respectively. A checkbox that controls the value of myglobalvar (.signals:one) is also added.

```
view signals
set myglobalvar(.signals:one) 0
set myglobalvar(.signals:two) 1
proc AddMyMenus {wname} {

  global myglobalvar
  set cmd1 "echo my_own_thing $wname"
  set cmd2 "echo my_to_upper  $wname"
  set cmd3 "echo my_to_lower  $wname"

  view signals
  set myglobalvar(.signals:one) 0
  set myglobalvar(.signals:two) 1
  proc AddMyMenus {wname} {
    global myglobalvar
    set cmd1 "echo my_own_thing $wname"
    set cmd2 "echo my_to_upper  $wname"
    set cmd3 "echo my_to_lower  $wname"
  }
```

ModelSim SE Command Reference
<table>
<thead>
<tr>
<th>#</th>
<th>WindowName</th>
<th>Menu</th>
<th>MenuItem label</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_menu</td>
<td>$wname</td>
<td>mine</td>
<td>0;# 0th letter (M) is underlined</td>
<td></td>
</tr>
<tr>
<td>add_menuitem</td>
<td>$wname</td>
<td>mine</td>
<td>&quot;Do My Own Thing...&quot;</td>
<td>$cmd1</td>
</tr>
<tr>
<td>add_separator</td>
<td>$wname</td>
<td>mine</td>
<td>;#----------------------------</td>
<td></td>
</tr>
<tr>
<td>add_submenu</td>
<td>$wname</td>
<td>mine</td>
<td>changeCase</td>
<td></td>
</tr>
<tr>
<td>add_menuitem</td>
<td>$wname</td>
<td>mine.changeCase</td>
<td>&quot;To Upper&quot;</td>
<td>$cmd2</td>
</tr>
<tr>
<td>add_menuitem</td>
<td>$wname</td>
<td>mine.changeCase</td>
<td>&quot;To Lower&quot;</td>
<td>$cmd3</td>
</tr>
<tr>
<td>add_submenu</td>
<td>$wname</td>
<td>mine</td>
<td>vars</td>
<td></td>
</tr>
<tr>
<td>add_menucb</td>
<td>$wname</td>
<td>mine.vars</td>
<td>&quot;Feature One&quot;</td>
<td>-variable myglobalvar($wname:one) -onvalue 1 -offvalue 0 -indicatoron 1</td>
</tr>
</tbody>
</table>

This example is available in the following DO file: <install_dir>/modeltech/examples/addmenu.do. You can run the DO file to add the "Mine" menu shown in the illustration, or modify the file for different results.

To execute the DO file, select **Tools > Execute Macro** (Main window), or use the **do** command (CR-138).

**See also**

- `add_menucb` (CR-53), `add_menuitem` (CR-54), `add_separator` (CR-55), `add_submenu` (CR-56), `change_menu_cmd` (CR-73)
add_menucb

The **add_menucb** command creates a checkbox within the specified menu of the specified window. A checkbox is a small box with a label. Clicking on the box will toggle the state, from on to off or the reverse. When the box is "on", the Tcl global variable `<var>` is set to `<onval>`. When the box is "off", the global variable is set to `<offval>`. Also, if something else changes the global variable, its current state is reflected in the state of the checkbox. Returns nothing.

**Syntax**

```
add_menucb
  <window_name>  <menu_name>  <Text>  -variable <var>  -onvalue <onval>
  -offvalue <offval>  [-indicatoron <val>]
```

**Arguments**

- `<window_name>`: Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).
- `<menu_name>`: Name of the Tk menu widget. Required.
- `<Text>`: Text to be displayed next to the checkbox. Required.
- `-variable <var>`: Global Tcl variable to be reflected and changed. Required.
- `-onvalue <onval>`: Value to set the global Tcl variable to when the box is "on". Required.
- `-offvalue <offval>`: Value to set the global Tcl variable to when the box is "off". Required.
- `-indicatoron <val>`: 0 or 1. If 1, the status indicator is displayed. Otherwise it is not displayed. Optional. The default is 1.

**Examples**

```
add_menucb $wname mine.vars "Feature One" -variable myglobalvar($wname:one) \
  -onvalue 1 -offvalue 0 -indicatoron 1
```

**See also**

- **add_menu** (CR-51), **add_menuitem** (CR-54), **add_separator** (CR-55), **add_submenu** (CR-56), **change_menu_cmd** (CR-73)

The add_menucb command is also used as part of the add_menu (CR-51) example.
add_menuitem

The add_menuitem command creates a menu item within the specified menu of the specified window. May be used within a submenu. Returns nothing.

Syntax

add_menuitem
  <window_name>  <menu_path>  <Text>  <Cmd>  [shortcut]

Arguments

<window_name>
  Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

<menu_path>
  Name of the Tk menu widget plus submenu path. Required.

<Text>
  Text to be displayed. Required.

<Cmd>
  The command to be executed when the menu item is selected with the left mouse button. To echo the command and display the return value in the Main window, prefix the command with the transcribe command (CR-228). Transcribe will also echo the results to the transcript window. Required.

<shortcut>
  Number of the letter in the menu name that is to be used as the shortcut. Numbering starts with 0 (i.e., first letter = 0, second letter = 1, third letter = 2, etc.). Optional. Default is "-1", which indicates no shortcut is to be used.

Examples

add_menuitem $wname user "Save Results As..." $my_save_cmd

See also

add_menu (CR-51), add_menucb (CR-53), add_separator (CR-55), add_submenu (CR-56), change_menu_cmd (CR-73)

The add_menuitem command is also used as part of the add_menu (CR-51) example.
add_separator

The **add_separator** command adds a separator as the next item in the specified menu path in the specified window. Returns nothing.

Syntax

```
add_separator  
<window_name>  <menu_path>
```

Arguments

<window_name>
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

<menu_path>
Name of the Tk menu widget plus submenu path. Required.

Examples

```
add_separator $wname user
```

See also

`add_menu (CR-51), add_menucb (CR-53), add_menuitem (CR-54), add_submenu (CR-56), change_menu_cmd (CR-73)`

The **add_separator** command is also used as part of the **add_menu** (CR-51) example.
add_submenu

The **add_submenu** command creates a cascading submenu within the specified menu_path of the specified window. May be used within a submenu.

Returns the full Tk path to the new submenu widget.

**Syntax**

```
add_submenu
<window_name> <menu_path> <name> [shortcut]
```

**Arguments**

- `<window_name>`
  Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

- `<menu_path>`
  Name of the Tk menu widget plus submenu path. Required.

- `<name>`
  Name to be displayed on the submenu. Required.

- `<shortcut>`
  Number of the letter in the menu name that is to be used as the shortcut. Numbering starts with 0 (i.e., first letter = 0, second letter = 1, third letter = 2, etc.). Optional. Default is "-1", which indicates no shortcut is to be used.

**See also**

add_menu (CR-51), add_menucb (CR-53), add_menuitem (CR-54), add_separator (CR-55), change_menu_cmd (CR-73)

The **add_submenu** command is also used as part of the **add_menu** (CR-51) example.
add wave

The add wave command adds VHDL signals and variables and Verilog nets and registers to the Wave window. It also allows specification of user-defined buses.

If no port mode is specified, add wave will display all items in the selected region with names matching the item name specification.

Limitations: VHDL variables and Verilog memories can be listed using the variable’s full name only (no wildcards).

Syntax

```
add wave
[-allowconstants] [-color <standard_color_name>] [-expand <signal_name>]
[-<format>] [-height <pixels>] [-in] [-inout] [-internal]
[-divider <divider_name>...] | [<item_name> | {[<item_name> [sig1 sig2 sig3 ...
...]]} ...] [-label <name>] [-noupdate] [-offset <offset>] [-out] [-ports]
[-<radix>] [-recursive] [-scale <scale>] [-window <wname>]
```

Arguments

- **-allowconstants**
  For use with wildcard searches. Specifies that constants matching the wildcard search should be added to the Wave window. Optional. By default, constants are ignored because they do not change.

- **-color <standard_color_name>**
  Specifies the color used to display a waveform. Optional. These are the standard X Window color names, or rgb value (e.g., #357f77); enclose 2-word names (“light blue”) in quotes.

- **-divider <divider_name>**
  Adds a divider with the specified name. Optional. You can specify one or more names. All names listed after -divider are taken to be names.

- **-expand <signal_name>**
  Causes a compound signal to be expanded immediately, but only one level down. Optional. The <signal_name> is required, and may include wildcards.

- **-<format>**
  Specifies the display format of the items:
  ```
literal
logic
analog-step
analog-interpolated
analog-backstep
```
  Optional. Literal waveforms are displayed as a box containing the item value. Logic signals may be U, X, 0, 1, Z, W, L, H, or '-'.

  The way each state is displayed is specified by the logic type display preferences, see "Preference variables located in INI files" (UM-444). Analog signals are sized by -scale and by -offset. Analog-step changes to the new time before plotting the new Y. Analog-interpolated draws a diagonal line. Analog-backstep plots the new Y before...
moving to the new time. See "Editing and formatting HDL items in the Wave window" (UM-261).

- `<height <pixels>>`
  Specifies the height (in pixels) of the waveform. Optional.

- `<in>`
  For use with wildcard searches. Specifies that the scope of the search is to include ports of mode IN if they match the item_name specification. Optional.

- `<inout>`
  For use with wildcard searches. Specifies that the scope of the search is to include ports of mode INOUT if they match the item_name specification. Optional.

- `<internal>`
  For use with wildcard searches. Specifies that the scope of the search is to include internal items (non-port items) if they match the item_name specification. Optional.

- `<item_name>`
  Specifies the names of HDL items to be included in the Wave window display. Optional. Wildcard characters are allowed. Note that the WildcardFilter Tcl preference variable identifies types to ignore when matching items with wildcard patterns. Variables may be added if preceded by the process name. For example,
  
  ```tcl
  add wave myproc/int1
  {<item_name> {<sig1> <sig2> <sig3> ...}}
  ```
  
  Creates a user-defined bus with the name `<item_name>`; `'<sigi>'` are signals to be concatenated within the user-defined bus. Optional. The following option is available:

  ▶ **Note:** You can also select **Tools > Combine Signals** (Wave window) to create a user-defined bus.

- `<label <name>>`
  Specifies an alternative name for the signal being added to the Wave window. Optional. For example,
  
  ```tcl
  add wave -label c clock
  ```
  
  adds the `clock` signal, labeled as "c", to the Wave window.

  This alternative name is not valid in a `force` (CR-156) or `examine` (CR-149) command; however, it can be used in a `search` command (CR-212) with the `wave` option.

- `<noupdate>`
  Prevents the Wave window from updating when a series of `add wave` commands are executed in series. Optional.

- `<offset <offset>>`
  Modifies an analog waveform’s position on the display. Optional. The offset value is part of the wave positioning equation (see `-scale` below).

- `<out>`
  For use with wildcard searches. Specifies that the scope of the search is to include ports of mode OUT if they match the item_name specification. Optional.
-ports
For use with wildcard searches. Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT. Optional.

-<radix>
Specifies the radix for the items that follow in the command. Optional. Valid entries (or any unique abbreviation) are:

- binary
- octal
- decimal (or signed)
- unsigned
- hexadecimal
- ascii
- symbolic
- default

If no radix is specified for an enumerated type, the default representation is used. You can change the default radix for the current simulation using the `radix` command (CR-200). You can change the default radix permanently by editing the `DefaultRadix` (UM-447) variable in the `modelsim.ini` file.

If you specify a radix for an array of a VHDL enumerated type, ModelSim converts each signal value to 1, 0, Z, or X. See also, "Preference variables located in Tcl files" (UM-454).

-recursive
For use with wildcard searches. Specifies that the scope of the search is to descend recursively into subregions. Optional; if omitted, the search is limited to the selected region.

-scale <scale>
Scales analog waveforms. Optional. The scale value is part of the wave positioning equation shown below.

The position and size of the waveform is given by:

\[(signal\_value + \text{<offset>}) \times \text{<scale}>\]

If \(signal\_value + \text{<offset>} = 0\), the waveform will be aligned with its name. The \text{<scale>} value determines the height of the waveform, 0 being a flat line.

-window <wname>
Adds HDL items to the specified window \text{<wname>} (e.g., wave2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the `view` command (CR-263) with the -new option to create a new window.
**Examples**

```
add wave -logic -color gold out2
   Displays an item named `out2`. The item is specified as being a logic item presented in gold.

add wave -hex {address {a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0}}
   Displays a user-defined, hex formatted bus named `address`.

add wave -r /*
   Waves all items in the design.

add wave *
   Waves all items in the region.

add wave -in *
   Waves all input ports in the region.

add wave -hex {mybus {scalar1 vector1 scalar2}}
   Creates a user-defined bus named "mybus" consisting of three signals. `scalar1` and `scalar2` are of type `std_logic` and `vector1` is of type `std_logic_vector (7 downto 1)`. The bus is displayed in hex.

   Slices and arrays may be added to the bus using either VHDL or Verilog syntax. For example:
   ```
   add wave (vector3(1))
   add wave (vector3[1])
   add wave (vector3(4 downto 0))
   add wave (vector3[4:0])
   ```

add wave vec1 -hex vec2 -dec vec3 vec4
   Adds the item `vec1` to the Wave window using symbolic values, adds `vec2` in hexadecimal, and adds `vec3` and `vec4` in decimal.
```

**See also**

```
add list (CR-48), log (CR-166), "Extended identifiers" (CR-15), "Concatenation directives" (CR-19)
```
The **alias** command displays or creates user-defined aliases. Any arguments passed on invocation of the alias will be passed through to the specified commands. Returns nothing. Existing ModelSim commands (e.g., run, env, etc.) cannot be aliased.

### Syntax

```bash
alias [ <name> ["<cmds>"]]
```

### Arguments

- `<name>`
  - Specifies the new procedure name to be used when invoking the commands.

- `"<cmds>"`
  - Specifies the command or commands to be evaluated when the alias is invoked.

### Examples

```bash
alias
Lists all aliases currently defined.

alias <name>
Lists the alias definition for the specified name if one exists.

alias <name>
Lists the alias definition for the specified name if one exists.

alias myquit "write list ./mylist.save; quit -f"
Creates a Tcl procedure, "myquit", that when executed, writes the contents of the List window to the file *mylist.save* by invoking `write list` (CR-325), and quits ModelSim by invoking `quit` (CR-199).```
batch_mode

The **batch_mode** command returns a 1 if ModelSim is operating in batch mode, otherwise it returns a 0. It is typically used as a condition in an if statement.

Syntax

```
batch_mode
```

Arguments

None

Examples

Some GUI commands do not exist in batch mode. If you want to write a script that will work in or out of batch mode, you can use the **batch_mode** command to determine which command to use. For example:

```
if [batch_mode] {
    log /*
} else {
    add wave /*
}
```

See also

"Running command-line and batch-mode simulations" (UM-490)
**bd**

The **bd** command deletes a breakpoint. You must specify a filename and line number, or a specific breakpoint id#. Multiple filename/line number pairs and id#s may be specified.

**Syntax**

```
bd
  <filename> <line_number> | <id#>
```

**Arguments**

- `<filename>`
  Specifies the name of the source file in which the breakpoint is to be deleted. Required if an id# is not specified. The filename must match the one used previously to set the breakpoint, including whether a full pathname or a relative name was used.

- `<line_number>`
  Specifies the line number of the breakpoint to be deleted. Required if an id# is not specified.

- `<id#>`
  Specifies the id number of the breakpoint to be deleted. Required if a filename and line number are not specified.

**Examples**

```
bd alu.vhd 127
  Deletes the breakpoint at line 127 in the source file named `alu.vhd`.

bd 5
  Deletes the breakpoint with id# 5.

bd 6 alu.vhd 234
  Deletes the breakpoint with id# 6 and the breakpoint at line 234 in the source file named `alu.vhd`.
```

**See also**

- `bp` (CR-68), `onbreak` (CR-179)
**bookmark add wave**

The **bookmark add wave** command creates a named reference to a specific zoom range and scroll position in the specified Wave window. Bookmarks are saved in the wave format file and are restored when the format file is read (see **write format** command (CR-323)).

**Syntax**

```
bookmark add wave
   <label>  <zoomrange>  <topindex>  [-window <window_name>]
```

**Arguments**

- `<label>`
  Specifies the name for the bookmark. Required.

- `<zoomrange>`
  Specifies a list of two times with optional units. Required. These two times must be enclosed in braces ({} or quotation marks ("").

- `<topindex>`
  Specifies the vertical scroll position of the window. Required. The number identifies which item the window should be scrolled to. For example, specifying 20 means the Wave window will be scrolled down to show the 20th item.

- `-window <window_name>`
  Specifies the window to which the bookmark will be added. Optional. If this argument is omitted, the bookmark is added in the current default Wave window.

**Examples**

```
bookmark add wave foo {{10 ns} {1000 ns}} 20
```

Adds a bookmark named "foo" to the current default Wave window. The bookmark marks a zoom range from 10ns to 1000ns and a scroll position of the 20th item in the window.

**See also**

- **bookmark delete wave** (CR-65), **bookmark goto wave** (CR-66), **bookmark list wave** (CR-67), **write format** (CR-323)
**bookmark delete wave**

The **bookmark delete wave** command deletes bookmarks from the specified Wave window.

**Syntax**

```
bookmark delete wave
   <label> [-all] [-window <window_name>]
```

**Arguments**

- `<label>`
  Specifies the name of the bookmark to delete. Required unless the -all switch is used.

- `-all`
  Specifies that all bookmarks in the window be deleted. Optional.

- `-window <window_name>`
  Specifies the window from which bookmark(s) will be deleted. Optional. If this argument is omitted, bookmark(s) in the current default Wave window are deleted.

**Examples**

```
bookmark delete wave foo
   Deletes the bookmark named "foo" from the current default Wave window.

bookmark delete wave -all -window wave1
   Deletes all bookmarks from the Wave window named "wave1".
```

**See also**

- `bookmark add wave` (CR-64), `bookmark goto wave` (CR-66), `bookmark list wave` (CR-67),
- `write format` (CR-323)
**bookmark goto wave**

The **bookmark goto wave** command zooms and scrolls a Wave window using the specified bookmark.

**Syntax**

```
bookmark goto wave
  <label> [-window <window_name>]
```

**Arguments**

- `<label>`
  - Specifies the bookmark to go to. Required.
- `-window <window_name>`
  - Specifies the Wave window to which the bookmark applies. Optional. Bookmarks can be used only in the windows in which they were originally created.

**See also**

- `bookmark add wave` (CR-64), `bookmark delete wave` (CR-65), `bookmark list wave` (CR-67), `write format` (CR-323)
bookmark list wave

The **bookmark list wave** command displays a list of available bookmarks in the Main window transcript.

**Syntax**

```
bookmark list wave
  [-window <window_name>]
```

**Arguments**

- `-window <window_name>`
  Specifies the Wave window for which you want a list of bookmarks. Optional. If this argument is omitted, ModelSim lists the bookmarks for the current default Wave window.

**See also**

- `bookmark add wave` (CR-64), `bookmark delete wave` (CR-65), `bookmark goto wave` (CR-66), `write format` (CR-323)
bp

The `bp` or breakpoint command either sets a file-line breakpoint or returns a list of currently set breakpoints. A set breakpoint affects every instance in the design unless the `-inst <region>` argument is used.

Syntax

```
bp <filename> <line_number> [-id <id#>] [-inst <region>] [-disable] [-cond {{<condition_expression>}} | {{<command>...}}] | [-query <filename> [line_number]]
```

Arguments

- `<filename>`
  Specifies the name of the source file in which to set the breakpoint. Required.

- `<line_number>`
  Specifies the line number at which the breakpoint is to be set. Required.

- `-id <id#>`
  Attempts to assign this id number to the breakpoint. Optional. If the id number you specify is already used, ModelSim will return an error.

- `-inst <region>`
  Sets the breakpoint so it applies only to the specified region. Optional.

- `-disable`
  Sets the breakpoint in a disabled state. Optional. You can enable the breakpoint later using the `enablebp` command (CR-145). By default, breakpoints are enabled when they are set.

- `-cond {{<condition_expression>}}`
  Specifies condition(s) that determine whether the breakpoint is hit. Optional. If the condition is true, the simulation stops at the breakpoint. If false, the simulation bypasses the breakpoint.

The condition can be an expression with these operators:

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>==, =</td>
</tr>
<tr>
<td>not equal</td>
<td>!!=, /=</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;, AND</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
</tbody>
</table>

Note: Ids for breakpoints are assigned from the same pool as those used for the `when` command (CR-314). So, even if you haven’t used an id number for a breakpoint, it’s possible it is used for a `when` command.
The operands may be item names, `signame`event, or constants. Subexpressions in parentheses are permitted. The command will be executed when the expression is evaluated as TRUE or 1.

The formal BNF syntax is:

```plaintext
condition ::= Name | { expression }

expression ::= expression AND relation
| expression OR relation
| relation

relation ::= Name = Literal
| Name /= Literal
| Name ` EVENT
| ( expression )

Literal ::= '"char"' | "<bitstring>" | bitstring
```

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals; i.e., Name = Name is not possible.

```plaintext
{<command>...}
```

Specifies one or more commands that are to be executed at the breakpoint. Optional. Multiple commands must be separated by semicolons (:) or placed on multiple lines. The entire command must be placed in curly braces.

Any commands that follow a `run` (CR-210) or `step` (CR-222) command will be ignored. A `run` or `step` command terminates the breakpoint sequence. This applies if macros are used within the `bp` command string as well. A `restore` (CR-206) command should not be used.

If many commands are needed after the breakpoint, they can be placed in a macro file.

```plaintext
-query <filename> [line_number] [line_number]
```

Returns information about the breakpoints set in the specified file. The information returned varies depending on which arguments you specify. See the examples below for details.

### Examples

**bp**

Lists all existing breakpoints in the design, including the source file names, line numbers, breakpoint id#s, and any commands that have been assigned to breakpoints.

**bp alu.vhd 147**

Sets a breakpoint in the source file `alu.vhd` at line 147.

**bp alu.vhd 147 {do macro.do}**

Executes the `macro.do` macro file after the breakpoint.

**bp -disable test.vhd 22 {echo [exa var1]; echo [exa var2]}**

Sets a breakpoint at line 22 of the file `test.vhd` and examines the values of the two variables `var1` and `var2`. This breakpoint is initially disabled. It can be enabled with the `enablebp` command (CR-145).
bp test.vhd 14 {if {now /= 100} then {cont}}
Sets a breakpoint in every instantiation of the file `test.vhd` at line 14. When that breakpoint is executed, the command is run. This command causes the simulator to continue if the current simulation time is not 100.

bp -query testadd.vhd
Lists the line number and enabled/disabled status (1 = enabled, 0 = disabled) of all breakpoints in `testadd.vhd`.

bp -query testadd.vhd 48
Lists details about the breakpoint on line 48. The output comprises six pieces of information: the first item (0 or 1) designates whether a breakpoint exists on the line (1 = exists, 0 = doesn’t exist); the second item is always 1; the third item is the file name in the compiled source; the fourth item is the breakpoint line number; the fifth item is the breakpoint id; and the sixth item (0 or 1) designates whether the breakpoint is enabled (1) or disabled (0).

bp -query testadd.vhd 2 59
Lists all executable lines in `testadd.vhd` between lines 2 and 59.

▶ **Note:** Any breakpoints set in VHDL code and called by either resolution functions or functions that appear in a port map are ignored.

**See also**

cd

The Tcl cd command changes the ModelSim local directory to the specified directory. See the Tcl man pages (Help > Tcl Man Pages) for any cd command options. Returns nothing.

Syntax

```plaintext
cd
   [ <dir> ]
```

Arguments

<dir>
   The directory to which to change. Optional. If no directory is specified, ModelSim changes to your home directory.

Description

After you change the directory with cd, ModelSim continues to write the `vsim.wlf` file in the directory where the first add wave (CR-57), add list (CR-48) or log (CR-166) command was executed. After completing simulation of one design, you can use the cd command to change to a new design, then use the vsim command (CR-298) to load a new design.

Use the where command (CR-318) or the Tcl pwd command to confirm the current directory.

See also

where (CR-318), vsim (CR-298), and the Tcl man page for the cd, pwd and exec commands
change

The change command modifies the value of a VHDL variable or Verilog register variable.

Syntax

```change  <variable>  <value>```

Arguments

```<variable>```
Specifies the name of a variable. Required. The variable name must specify a scalar type or a one-dimensional array of character enumeration. You may also specify a record subelement, an indexed array, a sliced array, or a bit or slice of a register, as long as the type is one of the above.

```<value>```
Defines a value for the variable. Required. The specified value must be appropriate for the type of the variable.

Examples

```change count 16#FFFF```
Changes the value of the variable count to the hexadecimal value FFFF.

```change rega[16] 0```
Changes the value of rega that is specified by the index (i.e., 16).

```change foo[20:22] 011```
Changes the value of foo that is specified by the slice (i.e., 20:22).

See also

```force``` (CR-156)
change_menu_cmd

The `change_menu_cmd` command changes the command to be executed for a specified menu item label, in the specified menu, in the specified window. The menu_path and label must already exist for this command to function. Returns nothing.

Syntax

```
change_menu_cmd
    <window_name>  <menu_path>  <label>  <Cmd>
```

Arguments

- `<window_name>`
  Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.)

- `<menu_path>`
  Name of an existing Tk menu widget plus any submenu path. Required.

- `<label>`
  Current label on the menu item. Required.

- `<Cmd>`
  New Tcl command to be executed when selected. Required.

See also

- `add_menu` (CR-51), `add_menucb` (CR-53), `add_menuitem` (CR-54), `add_separator` (CR-55), `add_submenu` (CR-56)
check contention add

The **check contention add** command enables contention checking for the specified nodes. The allowed nodes are Verilog nets and VHDL signals of types std_logic and std_logic_vector. Any other node types and nodes that don’t have multiple drivers are silently ignored by the command.

**Syntax**

```
check contention add
```

**Arguments**

- `-r`
  Specifies that contention checking is enabled recursively into subregions. Optional. If omitted, contention check enabling is limited to the current region.

- `-in`
  Enables checking on nodes of mode IN. Optional.

- `-out`
  Enables checking on nodes of mode OUT. Optional.

- `-inout`
  Enables checking on nodes of mode INOUT. Optional.

- `-internal`
  Enables checking on internal items. Optional.

- `-ports`
  Enables checking on nodes of modes IN, OUT, or INOUT. Optional.

- `<node_name>`
  Enables checking for the named node(s). Required.

**See also**

"Bus contention checking" (UM-498)
check contention config

The check contention config command allows you to write checking messages to a file (messages display on your screen by default). You may also configure the contention time limit.

Syntax

check contention config
   [ -file <filename> ] [ -time <limit> ]

Arguments

-file <filename>
   Specifies a file to which to write contention messages. Optional. If this option is selected, the messages are not displayed to the screen.

-time <limit>
   Specifies a time limit that a node may be in contention. Optional. Contention is detected if a node is in contention for as long as or longer than the limit. The default limit is 0.

See also

"Bus contention checking" (UM-498)
check contention off

The **check contention off** command disables contention checking for the specified nodes.

Syntax

```
check contention off
```

Arguments

- **-all**
  - Disables contention checking for all nodes that have checking enabled. Optional.

- **-r**
  - Specifies that contention checking is disabled recursively into subregions. Optional. If omitted, contention check disabling is limited to the current region.

- **-in**
  - Disables checking on nodes of mode IN. Optional.

- **-out**
  - Disables checking on nodes of mode OUT. Optional.

- **-inout**
  - Disables checking on nodes of mode INOUT. Optional.

- **-internal**
  - Disables checking on internal items. Optional.

- **-ports**
  - Disables checking on nodes of modes IN, OUT, or INOUT. Optional.

- **<node_name>**
  - Disables checking for the named node(s). Required.

See also

"Bus contention checking" (UM-498)
check float add

The **check float add** command enables float checking for the specified nodes. The allowed nodes are Verilog nets and VHDL signals of type std_logic and std_logic_vector (other types are silently ignored).

Note that you can set a time limit (the default is zero) for float checking using the `-time <limit>` argument to the **check float config** command (CR-78). If you choose to modify the limit, you should do so prior to invoking any **check float add** commands.

**Syntax**

```
check float add
```

**Arguments**

- **-r**
  Specifies that float checking is enabled recursively into subregions. Optional. If omitted, float check enabling is limited to the current region.

- **-in**
  Enables checking on nodes of mode IN. Optional.

- **-out**
  Enables checking on nodes of mode OUT. Optional.

- **-inout**
  Enables checking on nodes of mode INOUT. Optional.

- **-internal**
  Enables checking on internal items. Optional.

- **-ports**
  Enables checking on nodes of modes IN, OUT, or INOUT. Optional.

- **<node_name>**
  Enables checking for the named node(s). Required.

**See also**

"Bus float checking" (UM-498)
check float config

The check float config command allows you to write checking messages to a file (messages display on your screen by default). You may also configure the float time limit.

Syntax

```
check float config
  [-file <filename>] [-time <limit>]
```

Arguments

- **-file <filename>**
  Specifies a file to which to write float messages. Optional. If this option is selected, the messages are not displayed to the screen.

- **-time <limit>**
  Specifies a time limit that a node may be floating. Optional. An error is detected if a node is floating for as long as or longer than the limit. The default limit is 0. Note that you should configure the time limit prior to invoking any check float add commands.

See also

"Bus float checking" (UM-498)
The `check float off` command disables float checking for the specified nodes.

**Syntax**

```
check float off
```

**Arguments**

- `-all`
  - Disables float checking for all nodes that have checking enabled. Optional.
- `-r`
  - Specifies that float checking is disabled recursively into subregions. Optional. If omitted, float check disabling is limited to the current region.
- `-in`
  - Disables checking on nodes of mode IN. Optional.
- `-out`
  - Disables checking on nodes of mode OUT. Optional.
- `-inout`
  - Disables checking on nodes of mode INOUT. Optional.
- `-internal`
  - Disables checking on internal items. Optional.
- `-ports`
  - Disables checking on nodes of modes IN, OUT, or INOUT. Optional.
- `<node_name>`
  - Disables checking for the named node(s). Required.

**See also**

"Bus float checking" (UM-498)
check stable off

The check stable off command disables stability checking. You may later enable it with check stable on (CR-81), and meanwhile, the clock cycle numbers and boundaries are still tracked.

Syntax

check stable off

Arguments

None.

See also

"Design stability checking" (UM-499)
check stable on

The **check stable on** command enables stability checking on the entire design. Design stability checking detects when circuit activity has not settled within a user-defined period for synchronous designs.

**Syntax**

```plaintext
check stable on
    [-file <filename>] [-period <time>] [-strobe <time>]
```

**Arguments**

- `-file <filename>`
  Specifies a file to which to write the error messages. If this option is selected, the messages are not displayed to the screen. Optional.

- `-period <time>`
  Specifies the clock period (which is assumed to begin at the time the **check stable on** command is issued). Optional. This option is required the first time you invoke the **check stable on** command. It is not required if you later enable checking after it was disabled with the **check stable off** command (CR-80).

- `-strobe <time>`
  Specifies the elapsed time within each clock cycle that the stability check is performed. Optional. The default strobe time is the period time. If the strobe time falls on a period boundary, then the check is actually performed one timestep earlier. Normally the strobe time is specified as less than or equal to the period, but if it is greater than the period, then the check will skip cycles.

**Examples**

```plaintext
check stable on -period "100 ps" -strobe "199 ps"
```

Performs a stability check 99 ps into each even numbered clock cycle (cycle numbers start at 1).

**See also**

"Design stability checking" (UM-499)
The `checkpoint` command saves the state of your simulation. The `checkpoint` command saves the simulation kernel state, the `vsim.wlf` file, the list of the HDL items shown in the List and Wave windows, the file pointer positions for files opened under VHDL and the Verilog `$fopen` system task, and the states of foreign architectures. Changes you made interactively while running `vsim` are not saved; for example, macros, virtual objects, command-line interface additions like user-defined commands, and states of graphical user interface windows are not saved. Also, toggle statistics (see the `toggle report` command (CR-226)) are not saved.

Once saved, a checkpoint file may be used with the `restore` command (CR-206) during the same simulation to restore the simulation to a previous state. A VSIM session may also be started with a checkpoint file by using the `vsim -restore` command (CR-298).

Compression of the checkpoint file is controlled either by the `CheckpointCompressMode` variable in the modelsim.ini file or by the `-nocompress` argument to `vsim`.

### Syntax

```plaintext
checkpoint <filename>
```

### Arguments

**<filename>**
Specifies the name of the checkpoint file. Required.

### See also

- `restore` (CR-206), `restart` (CR-204), `vsim` (CR-298), "The difference between checkpoint/restore and restarting" (UM-489)
compare add

The **compare add** command compares signals in a reference design against signals in a test design. You can specify whether to compare two signals, all signals in the region, or just ports or a subset of ports. Constant signals such as parameters and generics are ignored. See Chapter 11 - Waveform Comparison for a general overview of waveform comparisons.

**Syntax**

```plaintext
compare add
    -clock <name> [-help] [-label <label>] [-list] [-<mode>] [-nowin]
    [-rebuild] [-recursive] [<referencePath>] [<testPath>]
    [-separator <string>] [-tol <delay>] [-tolLead <delay>]
    [-tolTrail <delay>] [-verbose] [-wavepane <n>] [-wave]
    [-when [<expression>]] [-win <wname>]
```

**Arguments**

- **-clock <name>**
  Specifies the clock definition to use when sampling the specified regions. Required for a clocked comparison; not used for asynchronous comparisons.

- **-help**
  Lists the description and syntax for the compare add command in the Main window transcript. Optional.

- **-label <label>**
  Specifies a name for the comparison when it is displayed in the Wave window. Optional.

- **-list**
  Causes specified comparisons to be displayed in the default List window. Optional.

- **-<mode>**
  Specifies the mode of signal types that are compared. Optional. The actual values the option may take are -in, -out, -inout, -internal, -port, and -all. You can use more than one mode option in the same command.

- **-nowin**
  Specifies that compare signals shouldn’t be added to any window. Optional. By default, compare signals are added to the default Wave window. See -wave below.

- **-rebuild**
  Rebuilds a fragmented bus in the test design region and compares it with the corresponding bus in the reference design region. Optional. If a signal is found having the same name as the reference signal, the -rebuild option is ignored. When rebuilding the test signal, the name of the reference signal is used as the wildcard prefix.

- **-recursive**
  Specifies that signals should also be selected in all nested subregions, and subregions of those, etc. Optional.

- **<referencePath>**
  A full path to the reference signal, region, or glob expression. Optional. If not specified, ModelSim uses the top region of the reference dataset. If the reference path is a region or
glob expression, then the test path must be a region (or left blank). If the reference path is a signal, the test path can be a signal or a region.

<testPath>
A full path to the test signal or region. Optional if the test path is the same as the reference path except for the dataset name.

-separator <string>
Used with the -rebuild option. When a bus has been broken into bits (bit blasted) by a synthesis tool, ModelSim expects a separator between the base bus name and the bit indication. This option identifies that separator. The default is "_". For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc.

-tol <delay>
Specifies the maximum time a test signal edge is allowed to lead or trail a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

-tolLead <delay>
Specifies the maximum time a test signal edge is allowed to lead a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

-tolTrail <delay>
Specifies the maximum time a test signal edge is allowed to trail a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be placed in curly braces.

**Graphical representation of tolLead and tolTrail**

```
Reference Signal

Test Signal

tolLead  tolTrail
```

-verbose
Prints information in the Main window confirming the signals selected for comparison and any type conversions employed. Optional.

-wavepane <n>
Specifies the pane of the Wave window in which the differences will be viewed. Optional.

-wave
Specifies that compare signals be added automatically to the default Wave window. Optional. Default.
-when {<expression>}
  Specifies a conditional expression that must evaluate to "true" or "1" for differences to be reported. Optional. The expression is evaluated at the start of an observed difference. See "GUI_expression_format" (CR-18) for legal expression syntax.

-win <wname>
  Specifies a particular window to which to add items. Optional. Used to specify a particular window when multiple instances of that window type exist.

Examples

compare add
  Selects signals in the reference and test dataset top region according to the default mode. Uses asynchronous comparison with the default tolerances. Assumes that the top regions of the reference and test datasets have the same name and contain the same signals with the same names.

compare add -port -clock myclock10 gold:.test_ringbuf.ring_inst
  Selects port signals of instance .test_ringbuf.ring_inst in both datasets to be compared and sampled on strobe myclock10.

compare add -all -r gold:/top/cpu test:/testbench/cpu
  Selects all signals in the cpu region to be compared asynchronously using the default tolerances. Requires that the reference and test relative hierarchies and signal names within the cpu region be identical, but they need not be the same above the cpu region.

compare add -clock clock12 gold:.top.s1
  Specifies that signal gold:.top.s1 should be sampled at clock12 and compared with test:.top.s1, also sampled at clock12.

compare add -tolLead {3 ns} -tolTrail {5 ns} gold:/asynch/abc/s1 sim:/flat/sigabc
  Specifies that signal gold:/asynch/abc/s1 should be compared asynchronously with signal sim:/flat/sigabc using a leading tolerance of 3 ns and a trailing tolerance of 5 ns.

compare add -rebuild gold:.counter1.count test:.counter2.cnt
  Causes signals test:.counter2.cnt_dd to be rebuilt into bus test:.counter2.cnt[...] and compared against gold:.counter1.count.

See also

compare annotate (CR-86), compare clock (CR-87), compare configure (CR-89), compare continue (CR-90), compare delete (CR-91), compare end (CR-92), compare info (CR-93), compare list (CR-95), compare options (CR-96), compare reload (CR-99), compare reset (CR-100), compare run (CR-101), compare savediffs (CR-102), compare saverules (CR-103), compare see (CR-104), compare start (CR-106), compare stop (CR-108), compare update (CR-109), and Chapter 11 - Waveform Comparison
compare annotate

The compare annotate command either flags a comparison difference as "ignore" or adds a text string annotation to the difference. The text string appears when the difference is viewed in error message info popups or in the output of a compare info command (CR-93).

Syntax

```
compare annotate
    [-ignore] [-noignore] [-text <message>] <idNum1> [ <idNum2> ... ]
```

Arguments

- **-ignore**
  Flags the specified difference as "ignore." Optional.

- **-noignore**
  Undoes a previous -ignore command. Optional.

- **-text <message>**
  Adds a text string annotation to the difference that is shown wherever the difference is viewed. Optional.

- **<idNum1>**
  Identifies the difference number to annotate. Required. You can obtain a difference’s number using the compare start command (CR-106) or a popup dialog. Difference numbers are ordered by time of the difference start, but there may be more than one difference starting at a given time.

- **<idNum2>...**
  Identifies a second, third, etc. difference number to be annotated in the same way as idNum1. Optional. These are individual references; ranges of numbers cannot be specified.

Examples

```
compare annotate -ignore diff0001 diff0002 diff0010
    Flags difference numbers 1, 2, and 10 as "ignore."

compare annotate -text "THIS IS A CRITICAL PROBLEM" diff0012
    Annotates difference number 12 with the message "THIS IS A CRITICAL PROBLEM."
```

See also

compare add (CR-83), compare info (CR-93), and Chapter 11 - Waveform Comparison
**compare clock**

The **compare clock** command defines a clock that can then be used for clocked-mode comparisons. In clocked-mode comparisons, signals are sampled and compared on a specified strobe.

**Syntax**

```
compare clock
   [-rising | -falling | -both] [-delete] [-offset <delay>]
   [-when {<expression>}] [-wavewindow <name>] [-wavepane <n>] <clock_name>
   <signal_path>
```

**Arguments**

- **-rising**
  Specifies that the rising edge of the specified signal should be used. Optional. This is the default.

- **-falling**
  Specifies that the falling edge of the specified signal should be used. Optional. The default is rising.

- **-both**
  Specifies that both the rising and the falling edge of the specified signal should be used. Optional. The default is rising.

- **-delete**
  Deletes an existing compare clock. Optional.

- **-offset <delay>**
  Specifies a time value for delaying the sample time beyond the specified signal edge. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

- **-when {<expression>}**
  Specifies a conditional expression that must evaluate to "true" or "1" for that clock edge to be used as a strobe. Optional. The expression is evaluated at the time of the clock edge, rather than after the delay has been applied. See "GUI_expression_format" (CR-18) for legal expression syntax.

- **-wavewindow <name>**
  Specifies the name of the Wave window in which the strobe will be viewed. Optional.

- **-wavepane <n>**
  Specifies the pane of the Wave window in which the strobe will be viewed. Optional.

- **<clock_name>**
  A name for this clock definition. Required. This name will be used with the compare add command when doing a clocked-mode comparison.

- **<signal_path>**
  A full path to the signal whose edges are to be used as the strobe trigger. Required.
Examples

```bash
compare clock -rising strobe gold:.top.clock
    Defines a clocked compare strobe named "strobe" that samples signals on the rising edge of signal gold:.top.clock.

compare clock -rising -delay {12 ns} clock12 gold:/mydesign/clka
    Defines a clocked compare strobe named "clock12" that samples signals 12 ns after the rising edge of signal gold:/mydesign/clka.
```

See also

```bash
compare add (CR-83), Chapter 11 - Waveform Comparison
```
**compare configure**

The **compare configure** command modifies options for compare signals and regions. The modified options are applied to all items in the specified compare path.

**Syntax**

```plaintext
compare configure
```

**Arguments**

- **-clock <name>**
  Changes the strobe signal for the comparison. Optional. If the comparison is currently asynchronous, it will be changed to clocked. This switch may not be used with the -tol, -tolLead, and -tolTrail options.

- **-recursive**
  Specifies that signals should also be selected in all nested subregions, and subregions of those, etc. Optional.

- **-tol <delay>**
  Specifies the default maximum time the test signal edge is allowed to trail or lead the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be in curly braces.

- **-tolLead <delay>**
  Specifies the maximum time a test signal edge is allowed to lead a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

- **-tolTrail <delay>**
  Specifies the maximum time a test signal edge is allowed to trail a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be placed in curly braces.

- **-when {<expression>}**
  Specifies a conditional expression that must evaluate to "true" or "1" for differences to be reported. Optional. The expression is evaluated at the start of an observed difference. See "**GUI_expression_format**" (CR-18) for legal expression syntax.

- **<comparePath>**
  Identifies the path of a compare signal, region, or glob expression. Required.

**See also**

**compare add** (CR-83), *Chapter 11 - Waveform Comparison*
compare continue

This command is used to continue with comparison difference computations that were suspended using the `compare stop` button or Control-C. If the comparison was not suspended, `compare continue` has no effect.

Syntax

    compare continue

Arguments

None

See also

`compare stop` (CR-108), Chapter 11 - Waveform Comparison
compare delete

The compare delete command deletes a signal or region from the current open comparison.

Syntax

```bash
compare delete
    [-recursive] <objectPath>
```

Arguments

- `<objectPath>` Path in the reference design to the signal or region to be deleted. Required. The dataset prefix is not needed.

See also

compare add (CR-83), Chapter 11 - Waveform Comparison
**compare end**

The `compare end` command closes the active comparison without saving any information.

**Syntax**

```plaintext
compare end
```

**Arguments**

None

**See also**

`compare add` (CR-83), *Chapter 11 - Waveform Comparison*
compare info

The **compare info** command lists the results of the comparison in the Main window transcript. To save the information to a file, use the -write argument.

**Syntax**

```plaintext
compare info
   [-all] [-count] [<endNum>] [-primaryonly] [-signals] [-secondaryonly]
   [<startNum>] [-summary] [-write <filename>]
```

**Arguments**

- **-all**
  Lists all differences (even those marked as “ignore”) in the output. Optional. By default, ignored differences are not listed in the output of a compare info command.

- **-count**
  Returns the total number of primary differences found.

- **<endNum>**
  Specifies the difference number to end with. Optional. If omitted ModelSim ends the listing with the last difference.

- **-primaryonly**
  Lists only differences on individual bits, ignoring aggregate values such as a bus. Optional.

- **-signals**
  Returns a Tcl list of compare signal names that have at least one difference.

- **-secondaryonly**
  Lists only aggregate value differences such as a bus, ignoring the individual bits.

- **<startNum>**
  Specifies the difference number to start with. Optional. If omitted ModelSim starts the listing with the first difference.

- **-summary**
  Lists only summary information. Optional.

- **-write <filename>**
  Saves the summary information to <filename> rather than the Main window transcript. Optional.

**Examples**

```plaintext
compare info
   Lists all errors in the Main window transcript.

compare info -summary
   Lists only an error summary in the Main window transcript.

compare info -write myerrorfile 20 50
   Writes errors 20 through 50 to the file named "myerrorfile".
```
See also

**compare add** (CR-83), **compare annotate** (CR-86), *Chapter 11 - Waveform Comparison*
compare list

Displays in the Main window a list of all the compare add commands currently in effect.

Syntax

```plaintext
compare list
[-expand]
```

Arguments

- `-expand`
  Expands groups specified by the compare add command to individual signals. Optional.

See also

compare add (CR-83), Chapter 11 - Waveform Comparison
**compare options**

The `compare options` command sets defaults for various waveform comparison commands. Those defaults are used when other compare commands are invoked during the current session. To set defaults permanently, edit the appropriate PrefCompare() Tcl variable in the pref.tcl file (see "Preference variables located in Tcl files" (UM-454) for details).

If no arguments are used, compare options returns the current setting for all options. If one option is given that requires a value, and if that value is not given, compare options returns the current value of that option.

**Syntax**

```
compare options
    [-noignoreVlogStrengths] [-maxsignal <n>] [-maxtotal <n>] [-listwin
    <name>] [-<mode>] [-separator <string>] [-tol <delay>] [-tolLead <delay>]
    [-tolTrail <delay>] [-track] [-notrack] [-vhd1xmatches] [-vhd1zmatches]
    [-vlogxmatches] [-vlogzmatches] [-wavepane <n>] [-wavewin <name>]
```

**Arguments**

- `-addwave`
  Specifies that new comparison objects are added automatically to the Wave window. Optional. Default. You can specify that objects aren’t added automatically using the `-noaddwave` argument.

- `-hide`
  Hides all comparisons except those that have at least one difference. Optional.

- `-noaddwave`
  Specifies that new comparison objects are not added automatically to the Wave window. Optional. The default is to add comparison objects automatically.

- `-show`
  Shows all comparisons even if they don’t have any differences. Optional.

- `-ignoreVlogStrengths`
  Specifies that Verilog net strengths should be ignored when comparing two Verilog nets. Optional. Related Tcl variable is PrefCompare(defaultIgnoreVerilogStrengths).

- `-noignoreVlogStrengths`
  Specifies that Verilog net strengths should not be ignored when comparing two Verilog nets. Optional. Related Tcl variable is PrefCompare(defaultIgnoreVerilogStrengths).

- `-listwin <name>`
  Causes specified comparisons to be displayed in the specified List window. Optional.

- `-maxsignal <n>`
  Specifies an upper limit for the total differences encountered on any one signal. When that limit is reached, ModelSim stops computing differences on that signal. Optional. The default is 100. Related Tcl variable is PrefCompare(defaultMaxSignalErrors).
-maxtotal <n>
Specifies an upper limit for the total differences encountered. When that limit is reached, ModelSim stops computing differences. Optional. The default is 1000. Related Tcl variable is PrefCompare(defaultMaxTotalErrors).

-mode>
Specifies the default mode of signal types that are compared with the compare add command (CR-83). Optional. The actual values the option may take are -in, -out, -inout, -internal, -port, and -all. More than one mode option may be used in the same compare options command.

-separator <string>
Used with the -rebuild option of the compare add command (CR-83). When a bus has been broken into bits (bit blasted) by a synthesis tool, ModelSim expects a separator between the base bus name and the bit indication. This option identifies that separator. The default is ".". For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc. Optional. Related Tcl variable is PrefCompare(defaultRebuildSeparator).

tol <delay>
Specifies the default maximum time the test signal edge is allowed to trail or lead the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be in curly braces.

tolLead <delay>
Specifies the default maximum time the test signal edge is allowed to lead the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be in curly braces. Related Tcl variable is PrefCompare(defaultLeadTolerance).

tolTrail <delay>
Specifies the default maximum time the test signal edge is allowed to trail the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be in curly braces. Related Tcl variable is PrefCompare(defaultTrailTolerance).

Graphical representation of tolLead and tolTrail

Reference Signal

Test Signal

tolLead
tolTrail

-track
Specifies that the waveform comparison should track the current simulation. Optional. The differences will be updated at the end of each "run" command, so if you want to see differences soon after they occur, use many relatively short run commands. Related Tcl variable is PrefCompare(defaultTrackLiveSim).
-notrack
   Specifies that the waveform comparison should not track the current simulation.
   Optional. Related Tcl variable is PrefCompare(defaultTrackLiveSim).

-vhdlxmatches
   Specifies those VHDL signal states that should be considered a match with a std_logic
   "X". Optional. Related Tcl variable is PrefCompare(defaultVHDLXMatches).

-vhdlzmatches
   Specifies those VHDL signal states that should be considered a match with a std_logic
   "Z". Optional. Related Tcl variable is PrefCompare(defaultVHDLZMatches).

-vlogxmatches
   Specifies those Verilog signal states that should be considered a match with a Verilog
   "X". Optional. Related Tcl variable is PrefCompare(defaultVLOGXMatches).

-vlogzmatches
   Specifies those Verilog signal states that should be considered a match with a Verilog
   "Z". Optional. Related Tcl variable is PrefCompare(defaultVLOGZMatches).

-wavepane <n>
   Specifies the default pane of the Wave window in which compare differences will be
   viewed. Optional. Related Tcl variable is PrefCompare(defaultWavePane).

-wavewin <name>
   Specifies the default name of the Wave window in which compare differences will be
   viewed. Optional. Related Tcl variable is PrefCompare(defaultWaveWindow).

Examples

compare options
   Returns the current value of all options.

compare options -maxtotal 2000
   Sets the maxtotal option to 2000 differences.

compare options -maxtotal
   Returns the current value of the maxtotal option.

compare options -ignoreVlogStrengths
   Sets the option to ignore Verilog net strengths.

compare options -vlogxmatches XZ0
   Verilog X will now match X, Z, or 0.

compare options -vhdlx UXW-
   VHDL std_logic X will now match 'U', 'X', 'W', or '-'.

compare options -tolLead (300 ps)
   Sets the leading tolerance for asynchronous comparisons to 300 picoseconds.

compare options -tolTrail (250 ps)
   Sets the trailing tolerance for asynchronous comparisons to 250 picoseconds.

See also

compare add (CR-83), compare clock (CR-87), Chapter 11 - Waveform Comparison
**compare reload**

The `compare reload` command reloads comparison differences to allow their viewing without recomputation. Prior to invoking `compare reload`, you must open the relevant datasets with the same names that were used during the original comparison.

**Syntax**

```
compare reload
   <rulesFilename> <diffsFilename>
```

**Arguments**

- `<rulesFilename>`
  Specifies the name of the file that was previously saved using the "compare saverules" command. Required. Must be the first argument.

- `<diffsFilename>`
  Specifies the name of the file that was previously saved using the "compare savediffs" command. Required.

**See also**

- `compare add` (CR-83), `compare savediffs` (CR-102), `compare saverules` (CR-103), `compare run` (CR-101), `compare start` (CR-106), *Chapter 11 - Waveform Comparison*
compare reset

Clears the current compare differences, allowing another compare run command to be executed. Does not modify any of the compare options or any of the signals selected for comparison. This allows you to re-run the comparison with different options or with a modified signal list.

Syntax

compare reset

Arguments

None

See also

compare add (CR-83), compare run (CR-101), and Chapter 11 - Waveform Comparison
**compare run**

The `compare run` command runs the difference computation on the signals selected via a `compare add` command. Reports in the Main window the total number of errors found.

**Syntax**

```plaintext
compare run
[<startTime>] [<endTime>]
```

**Arguments**

- `<startTime>`
  Specifies when to start computing differences. Optional. Default is zero. If a unit (e.g., ps) is used with the time value, the time must be in curly braces. The default units are determined by the simulation resolution. (Default simulation resolution is nanoseconds. Simulation resolution can be changed with the `-t` argument of the `vsim` command (CR-298)).

- `<endTime>`
  Specifies when to end computing differences. Optional. Default is the end of the dataset simulation run that ends earliest. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

**Examples**

```plaintext
compare run
  Computes differences over the entire time range.

compare run {5.3 ns} {57 ms}
  Computes differences from 5.3 nanoseconds to 57 milliseconds.
```

**See also**

- `compare add` (CR-83), `compare end` (CR-92), `compare start` (CR-106), Chapter 11 - Waveform Comparison
**compare savediffs**

The **compare savediffs** command saves the comparison results to a file that can be reloaded later. To be able to reload the file later, you must also save the comparison setup using the **compare saverules** command.

**Syntax**

```plaintext
compare savediffs
```

**Arguments**

*<diffsFilename>*  
Specifies the name of the file to create. Required. To load the file at a later time, use the **compare reload** command (CR-99).

**See also**

**compare add** (CR-83), **compare reload** (CR-99), **compare saverules** (CR-103), *Chapter 11 - Waveform Comparison*
compare saverules

The `compare saverules` command saves the comparison setup information (or "rules") to a file that can be re-executed later. The command saves compare options, clock definitions, and region and signal selections.

Syntax

```
compare saverules
    [-expand] <rulesFilename>
```

Arguments

- `[-expand]`: Expands groups specified by the `compare add` (CR-83) command to individual signals. Optional. If you added a region with the `compare add` command and then deleted signals from that region, you must use the "-expand" argument or the rules will not reflect the signal deletions.

- `<rulesFilename>`: Specifies the name of the file to which you want to save the rules. Required. To load the file at a later time, use the `compare reload` command (CR-99).

See also

`compare add` (CR-83), `compare reload` (CR-99), `compare savediffs` (CR-102), Chapter 11 - Waveform Comparison
compare see

The **compare see** command displays the specified comparison difference in the Wave window using whatever horizontal and vertical scrolling are necessary. The signal containing the specified difference will be highlighted, and the active cursor will be positioned at the starting time of the difference.

**Syntax**

```
compare see
    [-wavepane <n>] [-wavewin <name>]
```

**Arguments**

- `-first`
  Shows the first difference, ordered by time. Optional. Performs the same action as the Find First Difference button in the Wave window.

- `-next`
  Shows the next difference (in time) after the currently selected difference. Optional. Performs the same action as the Find Next Difference button in the Wave window.

- `-nextanno`
  Shows the next annotated difference (in time) after the currently selected difference. Optional. Performs the same action as the Next Annotated Difference button in the Wave window.

- `-last`
  Shows the last difference, ordered by time. Optional. Performs the same action as the Find Last Difference button in the Wave window.

- `-previous`
  Shows the previous difference (in time) before the currently selected difference. Optional. Performs the same action as the Previous Difference button in the Wave window.

- `-prevanno`
  Shows the previous annotated difference (in time) before the currently selected difference. Optional. Performs the same action as the Previous Annotated Difference button in the Wave window.

- `-wavepane <n>`
  Specifies the pane of the Wave window in which the difference should be shown. Optional.

- `-wavewin <name>`
  Specifies the name of the Wave window in which the difference should be shown. Optional.
Examples

\texttt{compare see -first}
  Shows the earliest difference (in time) in the default Wave window.

\texttt{compare see -next}
  Shows the next difference (in time) in the default Wave window.

See also

\texttt{compare add} (CR-83), \texttt{compare run} (CR-101), \textit{Chapter 11 - Waveform Comparison}
**compare start**

The **compare start** command begins a new dataset comparison. The datasets that you’ll be comparing must already be open.

**Syntax**

```
compare start
    [-batch] [-hide] [-show] [-maxsignal <n>] [-maxtotal <n>]
    [-refDelay <delay>] [-testDelay <delay>] <reference_dataset>
    [<test_dataset>]
```

**Arguments**

- **-batch**
  Specifies that comparisons will not be automatically inserted into the wave window. Optional.

- **-hide**
  Hides all comparisons except those that have at least one difference. Optional.

- **-show**
  Shows all comparisons even if they don’t have any differences. Optional.

- **-maxsignal <n>**
  Specifies an upper limit for the total differences encountered on any one signal. When that limit is reached, ModelSim stops computing differences on that signal. Optional. The default limit is 100. You can change the default using the **compare options** command (CR-96) or by editing the PrefCompare(defaultMaxSignalErrors) variable in the pref.tcl file.

- **-maxtotal <n>**
  Specifies an upper limit for the total differences encountered. When that limit is reached, ModelSim stops computing differences. Optional. The default limit is 1000. You can change the default using the **compare options** command (CR-96) or by editing the PrefCompare(defaultMaxTotalErrors) variable in the pref.tcl file.

- **-refDelay <delay>**
  Delays the reference dataset relative to the test dataset. Optional. If <delay> contains a unit, it must be enclosed in curly braces. Delays are applied to signals specified with the **compare add** command (CR-83). For each signal compared, a delayed virtual signal is created with "_d" appended to the signal name, and these are the signals viewed in the wave window comparison objects. The delay is not applied to signals specified in compare "when" expressions.

- **-testDelay <delay>**
  Delays the test dataset relative to the reference dataset. Optional. If <delay> contains a unit, it must be enclosed in curly braces. Delays are applied to signals specified with the **compare add** command (CR-83). For each signal compared, a delayed virtual signal is created with "_d" appended to the signal name, and these are the signals viewed in the wave window comparison objects. The delay is not applied to signals specified in compare "when" expressions.
<reference_dataset>
   The dataset to be used as the comparison reference. Required.

<test_dataset>
   The dataset to be tested against the reference. Optional. If not specified, ModelSim uses the current simulation. The reference and test datasets may be the same.

Examples

compare start gold
   Begins a waveform comparison between a dataset named “gold” and the current simulation. Assumes the gold dataset was already opened.

dataset open gold_typ.wlf gold
dataset open bad_typ.wlf test
compare start -maxtotal 5000 -maxsignal 1000 gold test
   This command sequence opens two datasets and starts a comparison between the two using greater than default limits for total differences encountered.

See also

compare add (CR-83), compare options (CR-96), compare stop (CR-108), Chapter 11 - Waveform Comparison
**compare stop**

This command is used internally by the `compare stop` button to suspend comparison computations in progress. If a `compare run` execution has returned to the VSIM prompt, `compare stop` has no effect. Under Unix, entering a Control-C character in the window that invoked ModelSim has the same effect as `compare stop`.

**Syntax**

```
compare stop
```

**Arguments**

None

**See also**

`compare run` (CR-101), `compare start` (CR-106), *Chapter 11 - Waveform Comparison*
compare update

This command is primarily used internally to update the comparison differences when comparing a live simulation against a .wlf file. The **compare update** command is called automatically at the completion of each simulation run if the "-track" compare option is in effect.

The user can also call compare update periodically during a long simulation run to cause difference computations to catch up with the simulation. This command does nothing if the -track compare option was not in effect when the **compare run** command (CR-101) was executed.

**Syntax**

```plaintext
compare update
```

**Arguments**

None

**See also**

**compare run** (CR-101), *Chapter 11 - Waveform Comparison*
The `configure` (config) command invokes the List or Wave widget configure command for the current default List or Wave window. To change the default window, use the `view` command (CR-263).

**Syntax**

```bash
configure list|wave [-window <wname>] [option <value>]
[-delta [all | collapse | none]] [-gateduration [<duration_open>]]
[-gateexpr [<expression>]] [-usegating [<value>]]
[-strobeperiod [<period>]] [-strobefast [<start_time>]]
[-usesignaltriggers [<value>]] [-usestrobe [<value>]]
[-childrowmargin [<pixels>]] [-gridcolor [<color>]]
[-namecolwidth [<width>]] [-rowmargin [<pixels>]]
[-signal根源width [<value>]]
[-timecolor [<color>]] [-valuecolwidth [<width>]] [-vectorcolor [<color>]]
```

**Description**

The command works in three modes:

- without options or values it returns a list of all attributes and their current values
- with just an option argument (without a value) it returns the current value of that attribute
- with one or more option-value pairs it changes the values of the specified attributes to the new values

The returned information has five fields for each attribute:

- the command-line switch
- the Tk widget resource name
- the Tk class name
- the default value
- and the current value

**Arguments**

- `list|wave` Specifies either the List or Wave widget to configure. Required.
- `window <wname>` Specifies the name of the List or Wave window to target for the `configure` command. (The `view` command (CR-263) allows you to create more than one List or Wave window). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the `view` command (CR-263).
<option> <value>
-<option> <value>
  Specifies the window background color. Optional.
-<option> <value>
  Specifies the window foreground color. Optional.
-<option> <value>
  Specifies the window foreground color when selected. Optional.
-<option> <value>
  Specifies the window foreground color when selected. Optional.
-<option> <value>
  Specifies the font used in the widget. Optional.
-<option> <value>
  Specifies the height in pixels of each row. Optional.

Arguments, List window only
-<option> <value>
  The all option displays a new line for each time step on which items change; collapse
displays the final value for each time step; and none turns off the display of the delta
column. To use -<option>, -<option> must be set to 1 (on). Optional.
-<option> <value>
  The duration for gating to remain open beyond when -<option> (below) becomes false,
expressed in x number of timescale units. Extends gating beyond the back edge (the last
list row in which the expression evaluates to true). Optional. The default value for normal
synchronous gating is zero. If -<option> is set to a non-zero value, a simulation value
will be displayed after the gate expression becomes false (if you don’t want the values
displayed, set -<option> to zero).
-<option> <value>
  Specifies the expression for trigger gating. Optional. (Use the -<option> argument to
enable trigger gating.) The expression is evaluated when the List window would
normally have displayed a row of data. See the "GUI_expression_format" (CR-18) for
information on expression syntax.
-<option> <value>
  Enables triggers to be gated on (a value of 1) or off (a value of 0) by an overriding
expression. Default is off. Optional. (Use the -<option> argument to specify the
expression.) See "Setting List window display properties" (UM-211) for additional
information on using gating with triggers.
-<option> <value>
  Specifies the period of the list strobe. (When using a time unit, the time value and unit
must be placed in curly braces.) Optional.
-<option> <value>
  Specifies the start time of the list strobe. Optional.
-<option> <value>
  If 1, uses signals as triggers; if 0, not. Optional.
-usestrobe [<value>]
    If 1, uses the strobe to trigger; if 0, not. Optional.

Arguments, Wave window only

-childrowmargin [<pixels>]
    Specifies the distance in pixels between child signals. Optional.

-gridcolor [<color>]
    Specifies the background grid color; the default is grey50. Optional.

-namecolwidth [<width>]
    Specifies in pixels the width of the name column in the Wave window. Optional.

-rowmargin [<pixels>]
    Specifies the distance in pixels between top-level signals.

-signalnamewidth [<value>]
    Controls the number of hierarchical regions displayed as part of a signal name shown in
    the pathname pane of the Wave window. Optional. Default of 0 displays the full path. 1
    displays only the leaf path element, 2 displays the last two path elements, and so on.

-timecolor [<color>]
    Specifies the time axis color; the default is green. Optional.

-valuecolwidth [<width>]
    Specifies in pixels the width of the value column in the Wave window.

-vectorcolor [<color>]
    Specifies the vector waveform color; the default is #b3ffb3. Optional.

-waveselectcolor [<color>]
    Specifies the waveform color of a selected item.

-waveselectenable [<value>]
    Specifies whether the waveform highlights when an item is selected. The default of 1
    enables highlighting. 0 disables highlighting.

Note: To get a more readable listing of all attributes and current values, use the lecho
(CR-163) command, which pretty-prints a Tcl list.

Examples

config list -strobeperiod
    Displays the current value of the strobeperiod attribute.

config list -strobeperiod (50 ns) -strobeenable 0 -usestrobe 1
    Sets the strobe waveform and turns it on.

config wave -vectorcolor blue
    Sets the wave vector color to blue.

config wave -signalnamewidth 1
    Sets the display in the current Wave window to show only the leaf path of each signal.
See also

view (CR-263)
**context**

The **context** command provides several operations on a context’s name. The option you specify determines the operation.

**Syntax**

```
context dataset | exists | isInst | isNet | isProc | isVar | join | parent |
split | tail | type
<name>
```

**Arguments**

context dataset <name>
- Return the dataset name from the name.

context exists <name>
- Returns 1 if context name is valid, 0 otherwise.

context isInst <name>
- Returns 1 if context name is an instance pathname, 0 otherwise.

context isNet <name>
- Returns 1 if context name is a Signal or Net pathname, 0 otherwise.

context isProc <name>
- Returns 1 if context name is a Process pathname, 0 otherwise.

context isVar <name>
- Returns 1 if context name is a VHDL Variable pathname, 0 otherwise.

context join <name> <name> ...
- Takes one or more context names and combines them, using the correct path separator.

context parent <name>
- Returns the parent path of name by removing the tail (see context tail).

context path <name>
- Returns the pathname portion of name, removing the dataset name.

context split <name>
- Returns a list whose elements are the path components in name. The first element of the list will be the dataset name if one is present in name, including the dataset separator. For example:

  ```
  context split /foo/bar/baz
  returns
  / foo bar baz .
  ```
context tail <name>
    Returns all of the characters in name after the last path separator. If name contains no separators then returns name. Any trailing path separator is discarded.

context type <name>
    Returns a string giving the acc type of context name.

<string>
    Name of a context object or region. Required. Does not have to be a valid object name unless the specified option requires this (i.e., exists or isInst).
coverage clear

The **coverage clear** command is used to clear all coverage data obtained during previous run commands. After this command is executed all line number execution count data will be reset.

**Syntax**

```
coverage clear
```

**Arguments**

None.

**See also**

*coverage reload* (CR-121), *coverage report* (CR-122)
coverage exclude clear

The `coverage exclude clear` command unloads a currently loaded exclusion filter file. Exclusion filter files specify files and line numbers that you wish to exclude from Code Coverage statistics (see "Excluding lines and files" (UM-335) for more details).}

Syntax

`coverage exclude clear`

Arguments

None.

See also

`coverage reload` (CR-121), `coverage report` (CR-122)
coverage exclude disable

The `coverage exclude disable` command disables a currently loaded exclusion filter file. Exclusion filter files specify files and line numbers that you wish to exclude from Code Coverage statistics (see "Excluding lines and files" (UM-335) for more details).

Syntax

```
coverage exclude disable
```

Arguments

None.

See also

`coverage reload` (CR-121), `coverage report` (CR-122)
coverage exclude enable

The `coverage exclude enable` command enables a previously disabled exclusion filter file. Exclusion filter files specify files and line numbers that you wish to exclude from Code Coverage statistics (see "Excluding lines and files" (UM-335) for more details).

Syntax

```
coverage exclude enable
```

Arguments

None.

See also

`coverage reload` (CR-121), `coverage report` (CR-122)
coverage exclude load

The **coverage exclude load** command loads an exclusion filter file. Exclusion filter files specify files and line numbers that you wish to exclude from Code Coverage statistics (see "Excluding lines and files" (UM-335) for more details).

**Syntax**

```
coverage exclude load
   <filename>
```

**Arguments**

`<filename>`

Specifies the file name of the exclusion filter you wish to load. Required. See "Excluding lines and files" (UM-335) for filter file syntax.

**See also**

`coverage reload` (CR-121), `coverage report` (CR-122)
coverage reload

The **coverage reload** command is used to seed the coverage statistics with the output of a previous **coverage report** command. This allows you (for example) to gather statistics from multiple simulation runs into a single report.

**Syntax**

```
coverage reload
   <filename> [-incremental] [-keep]
```

**Arguments**

- `<filename>`
  Specifies the file containing data to reload. Required. This file should be the output of a previous **coverage report -lines** command.

- `--incremental`
  Adds loaded coverage data to current data. Optional. Without this switch, loading coverage data overwrites existing data.

- `--keep`
  By default, source files listed in the file being reloaded that do NOT exist in the current design will have their coverage data discarded. By specifying the -keep option, the data will be kept, even though it does not correspond to any file or line in the current design.

The **coverage reload** command allows the accumulation of coverage statistics for multiple simulation invocations.

By doing a **coverage report -lines** at the end of each simulation, and then a **coverage reload -keep** at the start of each subsequent invocation of the simulator, one can accumulate coverage data for a suite of different designs.

**See also**

**coverage clear** (CR-116), **coverage report** (CR-122)
coverage report

The `coverage report` command produces textual output of coverage statistics. (Select Tools > Source Coverage (Main window) to view this data more interactively.) You can choose from a variety of report output using the arguments listed below.

Syntax

```
coverage report
    [-file <filename>] [-excluded | -lines | -total | -zeroes]
```

Arguments

- `file <filename>`
  Specifies a file name for the report. Optional. Default is to write the report to the Main window.

- `excluded`
  Writes out the files and lines that are currently being excluded by the user from the coverage analysis. Optional. This is the same information that is shown in the "Excluded tab" (UM-332).

- `lines`
  Writes out the source file summary data and after each file it writes out the details for each executable line in the file. Optional. This is the most detailed report.

  If you are intending to use the `coverage reload` command, you should specify this switch.

- `total`
  Writes out a one line summary of the total files, lines, hits and overall percentage for the current analysis. Optional. Useful for tracking if anything has changed.

- `zeroes`
  Writes out a detailed report like the -lines option but only reports on the lines that do not have any coverage. Optional.

See also

`coverage clear` (CR-116), `coverage reload` (CR-121), `vsim` (CR-298) -coverage option
dataset alias

The dataset alias command assigns an additional name (alias) to a dataset. The dataset can then be referenced by that alias. A dataset can have any number of aliases, but all dataset names and aliases must be unique.

Syntax

dataset alias
  <dataset_name>  <alias_name>

Arguments

<dataset_name>
  Specifies the name of the dataset to which to assign the alias. Required.

<alias_name>
  Specifies the alias name to assign to the dataset. Optional. If you don’t specify an alias_name, ModelSim lists current aliases for the specified dataset_name.

See also

dataset list (CR-127), dataset open (CR-128), dataset save (CR-130)
dataset clear

The `dataset clear` command removes all event data from the current simulation WLF file while keeping all currently logged signals logged. Subsequent run commands will continue to accumulate data in the WLF file.

**Syntax**

dataset clear

**Example**

```
add wave *
run 100000ns
dataset clear
run 100000ns
```

Clears data in the WLF file from time 0ns to 100000ns, then logs data into the WLF file from time 100000ns to 200000ns.

**See also**

"WLF files (datasets)" (UM-154), log (CR-166)
dataset close

The **dataset close** command closes an active dataset. To open a dataset, use the **dataset open** command.

Syntax

```bash
dataset close  
  <logicalname>  |  [-all]
```

Arguments

<logicalname>
- Specifies the logical name of the dataset or alias you wish to close. Required if -all isn’t used.

- `-all`
  - Closes all open datasets including the simulation. Optional.

See also

**dataset open** (CR-128)
dataset info

The **dataset info** command reports a variety of information about a dataset.

**Syntax**

```
dataset info
<option> <dataset_name>
```

**Arguments**

```
<option>
Identifies what information you want reported. Required. Only one option per command
is allowed. The current options include:

name - Returns the actual name of the dataset. Useful for identifying the real dataset name
of an alias.

file - Returns the name of the WLF file associated with the dataset.

exists - Returns "1" if the dataset exists; "0" if it doesn’t.

<dataset_name>
Specifies the name of the dataset or alias for which you want information. Required.
```

**See also**

- **dataset alias** (CR-123), **dataset list** (CR-127), **dataset open** (CR-128)
dataset list

The `dataset list` command lists all active datasets.

Syntax

```
dataset list
   -long
```

Arguments

- `-long`
  Lists the filename corresponding to each dataset’s or alias’ logical name. Optional.

See also

- `dataset alias` (CR-123), `dataset save` (CR-130)
dataset open

The **dataset open** command opens a WLF file (representing a prior simulation) and assigns it the logical name that you specify. To close a dataset, use **dataset close**.

Syntax

```plaintext
dataset open
   <filename> [logicalname]
```

Arguments

- `<filename>`
  Specifies the WLF file to open as a view-mode dataset. Required.

- `<logicalname>`
  Specifies the logical name for the dataset. Optional. This is a prefix that will identify the dataset in the current session. By default the dataset prefix will be the name of the specified WLF file.

Examples

```plaintext
dataset open last.wlf test
```

Opens the dataset file `last.wlf` and assigns it the logical name `test`.

See also

- **dataset alias** (CR-123), **dataset list** (CR-127), **dataset save** (CR-130), **vsim** (CR-298) -view option
dataset rename

The `dataset rename` command changes the logical name of a dataset to the new name you specify.

**Syntax**

```
dataset rename <logicalname> <newlogicalname>
```

**Arguments**

- `<logicalname>`
  Specifies the existing logical name of the dataset. Required.

- `<newlogicalname>`
  Specifies the new logical name for the dataset. Required.

**Examples**

```
dataset rename test test2
```

Renames the dataset file "test" to "test2".

**See also**

`dataset alias` (CR-123), `dataset list` (CR-127), `dataset open` (CR-128)
dataset save

The dataset save command writes data from the current WLF file to a specified file. This lets you save simulation data while the simulation is still in progress.

Syntax

```plaintext
dataset save
  <logicalname> <newlogicalname>
```

Arguments

- `<datasetname>`
  Specifies the name of the dataset you want to save. Required.
- `<filename>`
  Specifies the name of the file to save. Required.

Examples

```plaintext
dataset save sim gold.wlf
```
Saves all current log data in the sim dataset to the file "gold.wlf".

See also

dataset snapshot (CR-131)
The **dataset snapshot** command saves data from the current WLF file (`vsim.wlf` by default) at a specified interval. This lets you take sequential or cumulative "snapshots" of your simulation data.

**Syntax**

```plaintext
dataset snapshot

```

**Arguments**

- **-dir <directory>**
  Specifies a directory into which the files should be saved. Optional. Default is to save into the directory where ModelSim is writing the current WLF file.

- **-disable**
  Turns snapshotting off. Optional. All other options are ignored if you specify `-disable`.

- **-enable**
  Turns snapshotting on. Optional. Default.

- **-file <filename>**
  Specifies the name of the file to save. Optional. Default is "vsim_snapshot". ".wlf" will be appended to the file and possibly an incrementing suffix if `-filemode` is set to "increment".

- **-filemode overwrite | increment**
  Specifies whether to overwrite the snapshot file each time a snapshot occurs. Optional. Default is "overwrite". If you specify "increment", a new file is created for each snapshot.
  An incrementing suffix (0 to n) is added to each new file (e.g., `vsim_snapshot_0.wlf`).

- **-mode cumulative | sequential**
  Specifies whether to keep all data from the time signals are first logged. Optional. Default is "cumulative". If you specify "sequential", the current WLF file is cleared every time a snapshot is taken. See the examples for further details.

- **-report**
  Lists current snapshot settings in the Main window transcript. Optional. All other options are ignored if you specify `-report`.

- **-reset**
  Resets values back to defaults. Optional. The behavior is to reset to default, then apply remainder of arguments on command line. See examples below. If specified by itself without any other arguments, -reset disables dataset snapshot.

- **-size <file size>**
  Specifies that a snapshot occurs based on WLF file size. You must specify either `-size` or `-time`. See examples below.

- **-time <simulation time>**
  Specifies that a snapshot occurs based on simulation time. You must specify either `-time` or `-size`. See examples below.
Examples

```bash
dataset snapshot -size 10
 Creates the file vsim_snapshot.wlf that is written to every time the current WLF file reaches a multiple of 10 MB (i.e., at 10 MB, 20 MB, 30 MB, etc.).

dataset snapshot -size 10 -mode sequential
 Similar to the previous example but in this case the current WLF file is cleared every time it reaches 10 MB.

dataset snapshot -time 1000000 -file gold.wlf -mode sequential -filemode increment
 Assuming simulator time units are ps, this command saves a file called "gold_n.wlf" every 1000000 ps. If you ran for 3000000 ps, you’d have three files: gold_0.wlf with data from 0 to 1000000 ps, gold_1.wlf with data from 1000001 to 2000000, and gold_2.wlf with data from 2000001 to 3000000.

Note: Because this example uses "sequential" mode, if you ran the simulation for 3500000 ps, the resulting vsim.wlf (the default log file) file will contain data only from 3000001 to 3500000 ps.

dataset snapshot -reset -time 10000
 Enables snapshotting with time=10000 and default mode (cumulative) and default filemode (overwrite).
```

See also

```bash
dataset save (CR-130)
```
delete

The delete command removes HDL items from either the List or Wave window.

Syntax

delete
   list|wave [-window <wname>] <item_name>

Arguments

list|wave
   Specifies the target window for the delete command. Required.

-window <wname>
   Specifies the name of the List or Wave window to target for the delete command (the view command (CR-263) allows you to create more than one List or Wave window). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the view command (CR-263).

=item_name>
   Specifies the name of an item. Required. Must match an item name used in an add list (CR-48) or add wave (CR-57) command. Multiple item names may be specified. Wildcard characters are allowed.

Examples

delete list -window list2 vec2
   Removes the item vec2 from the list2 window.

See also

add list (CR-48), add wave (CR-57), and "Wildcard characters" (CR-16)
describe

The `describe` command displays information about the specified HDL item. The description is displayed in the Main window (UM-173). The following kinds of items can be described:

- **VHDL**
  - signals, variables, and constants
- **Verilog**
  - nets and registers

All but VHDL variables and constants may be specified as hierarchical names. VHDL variables and constants can be described only when visible from the current process that is either selected in the Process window or is the currently executing process (at a breakpoint for example).

**Syntax**

```
describe
    <name>
```

**Arguments**

`<name>`

Specifies the name of an HDL item. Wildcards are accepted. Required.
disablebp

The `disablebp` command turns off breakpoints and `when` commands. To turn the breakpoints or when statements back on again, use the `enablebp` command.

Syntax

disablebp [ <id#> ]

Arguments

<id#>

Specifies a breakpoint or `when` command id to disable. No other breakpoints or `when` commands are affected. Optional.

See also

`bd` (CR-63), `bp` (CR-68), `enablebp` command (CR-145), `onbreak` (CR-179), `resume` (CR-207), `when` (CR-314)
disable_menu

The `disable_menu` command disables the specified menu within the specified window. The disabled menu will become grayed-out, and nonresponsive. Returns nothing.

Syntax

```plaintext
disable_menu
<window_name> <menu_path>
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

`<menu_path>`
Name of the Tk menu-widget path. Required.

Examples

```plaintext
disable_menu "" File
Disables the file menu of the Main window.

disable_menu .mywindow File
Disables the file menu of the mywindow window.
```

See also

`add_menu` (CR-51), `enable_menu` (CR-146)
disable_menuitem

The `disable_menuitem` command disables a specified menu item within the specified menu_path of the specified window. The menu item will become grayed-out, and nonresponsive. Returns nothing.

Syntax

```c
disable_menuitem
  <window_name> <menu_path> <label>
```

Arguments

- `<window_name>`
  Tk path of the window containing the menu. Required.
  Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

- `<menu_path>`
  Name of the Tk menu-widget path. The path may include a submenu as shown in the example below. Required.

- `<label>`
  Menu item text. Required.

Examples

```c
disable_menuitem .mywindow file.save "Save Results As..."
```

This command locates the mywindow window, and disables the Save Results As... menu item in the save submenu of the file menu.

See also

- `add_menuitem` (CR-54), `enable_menuitem` (CR-147)
do

The **do** command executes commands contained in a macro file. A macro file can have any name and extension. An error encountered during the execution of a macro file causes its execution to be interrupted, unless an **onerror** command (CR-181), **onbreak** command (CR-179), or the OnErrorDefaultAction Tcl variable has specified the **resume** command (CR-207).

**Syntax**

```plaintext
do <filename> [parameter_value]
```

**Arguments**

- `<filename>`
  Specifies the name of the macro file to be executed. Required. The name can be a pathname or a relative file name.

  Pathnames are relative to the current working directory if the **do** command is executed from the command line. If the **do** command is executed from another macro file, pathnames are relative to the directory of the calling macro file. This allows groups of macro files to be moved to another directory and still work.

- `<parameter_value>`
  Specifies values that are to be passed to the corresponding parameters $1 through $9 in the macro file. Optional. Multiple parameter values must be separated by spaces.

  If you want to make the parameters optional (i.e., specify fewer parameter values than the number of parameters actually used in the macro), you must use the **argc** (UM-456) simulator state variable in the macro. See "Making macro parameters optional" (UM-436).

  Note that there is no limit on the number of parameters that can be passed to macros, but only nine values are visible at one time. You can use the **shift** command (CR-217) to see the other parameters.

**Examples**

```plaintext
do macros/stimulus 100
   This command executes the file macros/stimulus, passing the parameter value 100 to $1 in the macro file.

do testfile design.vhd 127
   If the macro file testfile contains the line **bp** $1 $2, this command would place a breakpoint in the source file named design.vhd at line 127.
```

**See also**

See "Command-line mode" (UM-491). ModelSim can search for DO files based on the path list specified by the **DOPATH** (UM-441) environment variable. A DO file can also be called by **modelsim.ini** at startup (see "Using a startup file" (UM-452)). The Main window transcript can be saved as a macro (see the **write transcript** command (CR-328)).
**down**

The `down` command searches for signal transitions or values in the specified List window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a signal takes on a particular value, or an expression of multiple signals evaluates to true. See the `up` command (CR-231) for related functionality.

The procedure for using `down` includes three steps: click on the desired signal; click on the desired starting location; issue the `down` command. (The `settime` command (CR-216) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

**Syntax**

```
down
   [-expr {<expression>}] [-falling] [-noglitch] [-rising]
   [-value <sig_value>] [-window <wname>] [<n>]
```

**Arguments**

- `-expr {<expression>}`
  The List window will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced List window. A signal may be specified either by its full path or by the shortcut label displayed in the List window.
  See "GUI_expression_format" (CR-18) for the format of the expression. The expression must be placed within curly braces.

- `-falling`
  Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

- `-noglitch`
  Specifies that delta-width glitches are to be ignored. Optional.

- `-rising`
  Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

- `-value <sig_value>`
  Specifies a value of the signal to match. Optional. Must be specified in the same radix that the selected signal is displayed. Case is ignored, but otherwise the value must be an exact string match -- don’t-care bits are not yet implemented.

- `-window <wname>`
  Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the `view` command (CR-263) to change the default window.
<n>
Specifies to find the nth match. Optional. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.

**Examples**

down -noglitch -value FF23
   Finds the next time at which the selected vector transitions to FF23, ignoring glitches.

down
   Goes to the next transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-18) and can be built with the aid of the "The GUI Expression Builder" (UM-305).

down -expr {clk’rising && (mystate == reading) && (/top/u3/addr == 32’habcd1234)}
   Searches down for an expression that evaluates to a boolean 1 when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /top/u3/addr is equal to the specified 32-bit hex constant.

down -expr {(/top/u3/addr and 32’hff000000) == 32’hac000000}
   Searches down for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal /top/u3/adder equals hex ac.

down -expr {((NOW > 23 us) && (NOW < 54 us)) && clk’rising && (mode == writing)}
   Searches down for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal mode is enumeration writing.

**See also**

"GUI_expression_format" (CR-18), view (CR-263), seetime (CR-216), up (CR-231)
drivers

The `drivers` command displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net. The driver list is expressed relative to the top-most design signal/net connected to the specified signal/net. If the signal/net is a record or array, each subelement is displayed individually. This command reveals the operation of transport and inertial delays and assists in debugging models.

Syntax

```
drivers <item_name>
```

Arguments

`<item_name>`
- Specifies the name of the signal or net whose values are to be shown. Required. All signal or net types are valid. Multiple names and wildcards are accepted.
dumplog64

The **dumplog64** command dumps the contents of the specified WLF file in a readable format to stdout. The WLF file cannot be opened for writing in a simulation when you use this command.

The **dumplog64** command cannot be used in a DO file.

**Syntax**

```plaintext
dumplog64 <filename>
```

**Arguments**

```
<filename>
```

The name of the WLF file to be read. Required.
echo

The `echo` command displays a specified message in the Main window.

Syntax

```bash
echo
    [<text_string>]
```

Arguments

`<text_string>`

Specifies the message text to be displayed. Optional. If the text string is surrounded by quotes, blank spaces are displayed as entered. If quotes are omitted, two or more adjacent blank spaces are compressed into one space.

Examples

```bash
echo "The time is $now ns."
```

If the current time is 1000 ns, this command produces the message:

```
The time is     1000 ns.
```

If the quotes are omitted, all blank spaces of two or more are compressed into one space.

```bash
echo The time is $now ns.
```

If the current time is 1000ns, this command produces the message:

```
The time is 1000 ns.
```

`echo` can also use command substitution, such as:

```bash
echo The hex value of counter is `[examine -hex counter]`
```

If the current value of counter is 21 (15 hex), this command produces:

```
The hex value of counter is 15.
```
**edit**

The `edit` command invokes the editor specified by the EDITOR environment variable.

**Syntax**

```
edit
[<filename>]
```

**Arguments**

`<filename>`

Specifies the name of the file to edit. Optional. If the `<filename>` is omitted, the editor opens the current source file. If you specify a non-existent filename, it will open a new file.

**See also**

`notepad` (CR-176), and the `EDITOR` (UM-441) environment variable
enablebp

The `enablebp` command turns on breakpoints and `when` commands that were turned off by the `disablebp` command (CR-135).

Syntax

```
enablebp [<id#>]
```

Arguments

<id#>

Specifies a breakpoint or when statement id to enable. No other breakpoints or `when` commands are affected. Optional.

See also

`bd` (CR-63), `bp` (CR-68), `disablebp` command (CR-135), `onbreak` (CR-179), `resume` (CR-207), `when` (CR-314)
enable_menu

The `enable_menu` command enables a previously-disabled menu. The menu will be changed from grayed-out to normal and will become responsive. Returns nothing.

Syntax

```
enable_menu
    <window_name> <menu_path>
```

Arguments

- `<window_name>`
  Tk path of the window containing the menu. Required.
  Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

- `<menu_path>`
  Name of the Tk menu-widget path. Required.

Examples

```
enable_menu "" File
    Enables the previously-disabled file menu of the Main window.

enable_menu .mywindow File
    Enables the previously-disabled file menu of the mywindow window.
```

See also

- `add_menu` (CR-51), `disable_menu` (CR-136)
enable_menuitem

The `enable_menuitem` command enables a previously-disabled menu item. The menu item will be changed from grayed-out to normal, and will become responsive. Returns nothing.

Syntax

```bash
enable_menuitem <window_name> <menu_path> <label>
```

Arguments

- `<window_name>`
  Tk path of the window containing the menu. Required.
  Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

- `<menu_path>`
  Name of the Tk menu-widget path. The path may include a submenu as shown in the example below. Required.

- `<label>`
  Menu item text. Required.

Examples

```bash
enable_menuitem .mywindow file.save "Save Results As..."
```

This command locates the mywindow window, and enables the previously-disabled Save Results As... menu item in the save submenu of the file menu.

See also

`add_menuitem` (CR-54), `disable_menuitem` (CR-137)
environment

The `environment`, or `env` command, allows you to display or change the current dataset and region/signal environment.

Syntax

```
environment [-dataset] [-nodataset] [<dataset_prefix>[<pathname>]]
```

Arguments

- `-dataset`
  Displays the specified environment pathname with a dataset prefix. Optional. Dataset prefixes are displayed by default if more than one dataset is open during a simulation session.

- `-nodataset`
  Displays the specified environment pathname without a dataset prefix. Optional.

- `<dataset_prefix>`
  Changes all unlocked windows to the specified dataset context. Optional. The prefix is the logical name of the dataset followed by a colon (e.g., "sim:"). If the `<pathname>` argument is specified as well, it will change the environment to that specified context. If `<pathname>` is omitted, the environment reflects the previously set context.

- `<pathname>`
  Specifies the pathname to which the current region/signal environment is to be changed. Optional. If omitted the command causes the pathname of the current region/signal environment to be displayed.

  Multiple levels of a pathname must be separated by the character specified in the `PathSeparator` (UM-449). A single path separator character can be entered to indicate the top level. Two dots (..) can be entered to move up one level.

Examples

```
env
Displays the pathname of the current region/signal environment.

env -dataset test
Changes all unlocked windows to the context of the "test" dataset.

env test:/top/foo
Changes all unlocked windows to the context "test:/top/foo".

env blk1/u2
Moves down two levels in the design hierarchy.

env /
Moves to the top level of the design hierarchy.
```
examine

The examine, or exa command, examines one or more HDL items, and displays current values (or the values at a specified previous time) in the Main window (UM-173). It optionally can compute the value of an expression of one or more items.

The following items can be examined at any time:

- **VHDL**
  - signals, shared variables, process variables, and generics
- **Verilog**
  - nets and register variables

To display a previous value, specify the desired time using the `-time` option. To compute an expression, use the `-expr` option. The `-expr` and the `-time` options may be used together.

Virtual signals and functions may also be examined within the GUI (actual signals are examined in the kernel).

**Syntax**

```
```

**Arguments**

- `-delta <delta>`
  - Specifies a simulation cycle at the specified time from which to fetch the value. The default is to use the last delta of the time step. The items to be examined must be logged via the add list, add wave, or log command in order for the examine command to be able to return a value for a requested delta. Optional.

- `-env <path>`
  - Specifies a path to look for a signal name. Optional.

- `-expr <expression>`
  - Specifies an expression to be evaluated. Optional. The items to be examined must be logged via the add list, add wave, or log command in order for the examine command to be able to evaluate the specified expression. If the `-time` argument is present, the expression will be evaluated at the specified time, otherwise it will be evaluated at the current simulation time. See "GUI_expression_format" (CR-18) for the format of the expression. The expression must be placed within curly braces.

- `-in`
  - Specifies that `<name>` include ports of mode IN. Optional.

- `-out`
  - Specifies that `<name>` include ports of mode OUT. Optional.

- `-inout`
  - Specifies that `<name>` include ports of mode INOUT. Optional.

- `-internal`
  - Specifies that `<name>` include internal (non-port) signals. Optional.
-ports
  Specifies that `<name>` include all ports. Optional. Has the same effect as specifying -in,
  -inout, and -out together.

-name
  Displays signal name(s) along with the value(s). Optional. Default is -value behavior
  (see below).

  The lecho command (CR-163) will return the output of an examine command in
  pretty-print format. For example,
  lecho [examine -name clk prw pstrb]

-<radix>
  Specifies the radix for the items that follow in the command. Optional. Valid entries
  (including unique abbreviations) are:

  binary
  octal
  decimal (default for integers) or signed
  hexadecimal
  unsigned
  ascii
  symbolic
  default

  Entries may be truncated to any length, for example, -binary could be expressed as -b or
  -bin, etc. You can change the default radix for the current simulation using the radix
  command (CR-200). You can change the default radix permanently by editing the
  DefaultRadix (UM-447) variable in the modelsim.ini file.

-time <time>
  Specifies the time value between 0 and $now for which to examine the items. If an
  expression is specified it will be evaluated at that time. The items to be examined must
  be logged via the add list, add wave, or log command in order for the examine command
  to be able to return a value for a requested time. Optional.

  If the <time> field uses a unit, the value and unit must be placed in curly braces. For
  example, the following are equivalent for ps resolution:

  exa -time {3.6 ns} signal_a
  exa -time 3600 signal_a

-value
  Returns value(s) as a curly-braces separated Tcl list. Default. Use to toggle off a previous
  use of -name.

  <name>...
  Specifies the name of any HDL item. Required (except when the -expr option is used).
  All item types are allowed, except those of the type file. Multiple names and wildcards
  are accepted. To examine a VHDL variable you can add a process label to the name. For
  example (make certain to use two underscore characters):

  exa line__36/1
Examples

```
examine rega[16]
Returns the value of rega that is specified by the index (i.e., 16).
```

```
examine foo[20:22]
Returns the value of foo specified by the slice (i.e., 20:22).
```

```
examine -time {3450 us} -expr {/top/bus and $bit_mask}
In this example the -expr option specifies a signal path and user-defined Tcl variable.
The expression will be evaluated at 3450us.
```

```
examine -expr {clk'event && (/top/xyz == 16'hffae)}
Because -time is not specified, this expression will be evaluated at the current simulation time. Note the signal attribute and array constant specified in the expression.
```

Commands like `find` (CR-153) and `examine` return their results as a Tcl list (just a blank-separated list of strings). You can do things like:

```
foreach sig [find ABC*] {echo "Signal $sig is [exa $sig]" ...}
```

```
if {[examine -bin signal_12] == "11101111XXXZ"} {...}
```

```
examine -hex [find *]
```

**Note:** The Tcl variable array, $examine(), can also be used to return values. For example, $examine(/clk). You can also examine an item in the Source window (UM-229) by selecting it with the right mouse button.

See also

"GUI_expression_format" (CR-18)
exit

The `exit` command exits the simulator and the ModelSim application.

Syntax

```plaintext
exit
[-force]
```

Argument

`-force`
Quits without asking for confirmation. Optional; if this argument is omitted, ModelSim asks you for confirmation before exiting.

▶ Note: If you want to stop the simulation using a `when` command (CR-314), you must use a `stop` command (CR-223) within your when statement. DO NOT use an `exit` command or a `quit` command (CR-199). The `stop` command acts like a breakpoint at the time it is evaluated.
find

The `find` command locates items in the design whose names match the name specification you provide. You must specify the type of item you want to find. When searching for nets and signals, the `find` command returns the full pathname of all nets, signals, register variables, and named events that match the name specification.

When searching for nets and signals, the order in which arguments are specified is unimportant. When searching for virtuals, however, all optional arguments must be specified before any item names.

You can also use the `find` command to locate incrTcl classes and objects. See "incrTcl commands" in the Tcl Man Pages for more information.

Syntax

```
find nets | signals
   [-in] [-inout] [-internal] <item_name> ... [-nofilter] [-out] [-ports]
   [-recursive]

find virtuals
   [-kind <kind>] [-unsaved] <item_name> ...

find classes
   [<class_name>]

find objects
   [-class <class_name>] [-isa <class_name>] [<object_name>]
```

Arguments for nets and signals

- **-in**
  Specifies that the scope of the search is to include ports of mode IN. Optional.

- **-inout**
  Specifies that the scope of the search is to include ports of mode INOUT. Optional.

- **-internal**
  Specifies that the scope of the search is to include internal items. Optional.

- **<item_name> ...**
  Specifies the net or signal for which you want to search. Required. Multiple nets and signals and wildcard characters are allowed. Wildcard characters are accepted for primary names only. Wildcards in index and record filed names are not supported. Note that the `WildcardFilter` Tcl preference variable identifies types to ignore when matching items with wildcard characters unless you use the `-nofilter` argument.

- **-nofilter**
  Specifies that the `WildcardFilter` Tcl preference variable be ignored when finding signals or nets. Optional.

- **-out**
  Specifies that the scope of the search is to include ports of mode OUT. Optional.

- **-ports**
  Specifies that the scope of the search is to include all ports. Optional. Has the same effect as specifying `-in`, `-out`, and `-inout` together.
-recursive
Specifies that the scope of the search is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region.

Arguments for virtuals

-kind <kind>
Specifies the kind of virtual object for which you want to search. Optional. <kind> can be one of designs, explicits, functions, implicits, or signals.

-unsaved
Specifies that ModelSim find only virtuals that have not been saved to a format file.

<item_name> ...
Specifies the virtual object for which you want to search. Required. Multiple virtuals and wildcard characters are allowed.

Arguments for classes

<class_name>
Specifies the incrTcl class for which you want to search. Optional. Wildcard characters are allowed. The options for class_name include nets, objects, signals, and virtuals. If you do not specify a class name, the command returns all classes in the current namespace context. See "incrTcl commands" in the Tcl Man Pages for more information.

Arguments for objects

-class <class_name>
Restricts the search to objects whose most-specific class is class_name. Optional.

-isa <class_name>
Restricts the search to those objects that have class_name anywhere in their heritage. Optional.

<object_name>
Specifies the incrTcl object for which you want to search. Optional. Wildcard characters are allowed. If you do not specify an object name, the command returns all objects in the current namespace context. See "incrTcl commands" in the Tcl Man Pages for more information.

Examples

find signals -r /*
Finds all signals in the entire design.

find nets -in /top/xy*
Finds all input signals in region /top that begin with the letters "xy".

Additional search options

To search for HDL items within a specific display window, use the search command (CR-212) or select Edit > Find.
See also

"Wildcard characters" (CR-16)
force

The `force` command allows you to apply stimulus interactively to VHDL signals and Verilog nets and registers. Since `force` commands (like all commands) can be included in a macro file, it is possible to create complex sequences of stimuli.

You can force Virtual signals if the number of bits corresponds to the signal value. You cannot force virtual functions. In VHDL and mixed models, you cannot force an input port that is mapped at a higher level or that has a conversion function on the input.

You cannot force Verilog register variables – reg, integer, time, real (or realtime). These must be changed. See the `change` command.

Syntax

```
force [-freeze | -drive | -deposit] [-cancel <time>] [-repeat <time>] <item_name> <value> [<time>] [, <value> <time>] ...
```

Arguments

- `-freeze`
  Freezes the item at the specified value until it is forced again or until it is unforced with a `noforce` command (CR-173). Optional.

- `-drive`
  Attaches a driver to the item and drives the specified value until the item is forced again or until it is unforced with a `noforce` command (CR-173). Optional.
  This option is illegal for unresolved signals.

- `-deposit`
  Sets the item to the specified value. The value remains until there is a subsequent driver transaction, or until the item is forced again, or until it is unforced with a `noforce` command (CR-173). Optional.
  If one of the `-freeze`, `-drive`, or `-deposit` options is not used, then `-freeze` is the default for unresolved items and `-drive` is the default for resolved items.
  If you prefer `-freeze` as the default for resolved and unresolved VHDL signals, change the default force kind in the `DefaultForceKind` (UM-447) preference variable.

- `-cancel <time>`
  Cancels the `force` command at the specified `<time>`. The time is relative to the current time unless an absolute time is specified by preceding the value with the character @. Cancellation occurs at the last simulation delta cycle of a time unit. A value of zero cancels the force at the end of the current time period. Optional.

- `-repeat <time>`
  Repeats the `force` command, where `<time>` is the time at which to start repeating the cycle. The time is relative to the current time. A repeating `force` command will force a value before other non-repeating `force` commands that occur in the same time step. Optional.
<item_name>
Specifies the name of the HDL item to be forced. Required. A wildcard is permitted only if it matches one item. See "HDL item pathnames" (CR-14) for the full syntax of an item name. The item name must specify a scalar type or a one-dimensional array of character enumeration. You may also specify a record subelement, an indexed array, or a sliced array, as long as the type is one of the above. Required.

<value>
Specifies the value to which the item is to be forced. The specified value must be appropriate for the type. Required.

A VHDL one-dimensional array of character enumeration can be forced as a sequence of character literals or as a based number with a radix of 2, 8, 10 or 16. For example, the following values are equivalent for a signal of type bit_vector (0 to 3):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>character literal sequence</td>
</tr>
<tr>
<td>2#1111</td>
<td>binary radix</td>
</tr>
<tr>
<td>10#15</td>
<td>decimal radix</td>
</tr>
<tr>
<td>16#F</td>
<td>hexadecimal radix</td>
</tr>
</tbody>
</table>

**Note:** For based numbers in VHDL, ModelSim translates each 1 or 0 to the appropriate value for the number’s enumerated type. The translation is controlled by the translation table in the *pref.tcl* file. If ModelSim cannot find a translation for 0 or 1, it uses the left bound of the signal type (type'left) for that value.

<time>
Specifies the time to which the value is to be applied. The time is relative to the current time unless an absolute time is specified by preceding the value with the character @. If the time units are not specified, then the default is the resolution units selected at simulation start-up. Optional.

A zero-delay force command causes the change to occur in the current (rather than the next) simulation delta cycle.

### Examples

```
force input1 0
Forces input1 to 0 at the current simulator time.

force bus1 01XZ 100 ns
Forces bus1 to 01XZ at 100 nanoseconds after the current simulator time.

force bus1 16#F @200
Forces bus1 to 16#F at the absolute time 200 measured in the resolution units selected at simulation start-up.
```
force input1 1 10, 0 20 -r 100
Forces input1 to 1 at 10 time units after the current simulation time and to 0 at 20 time units after the current simulation time. This cycle repeats starting at 100 time units after the current simulation time, so the next transition is to 1 at 100 time units after the current simulation time.

force input1 1 10 ns, 0 {20 ns} -r 100ns
Similar to the previous example, but also specifies the time units. Time unit expressions preceding the "-r" must be placed in curly braces.

force s 1 0, 0 100 -repeat 200 -cancel 1000
Forces signal s to alternate between values 1 and 0 every 100 time units until time 1000. Cancellation occurs at the last simulation delta cycle of a time unit. So,

force s 1 0 -cancel 0
will force signal s to 1 for the duration of the current time period.

See also

noforce (CR-173), change (CR-72)

Note: You can configure defaults for the force command by setting the DefaultForceKind variable in the modelsim.ini file. See "Force command defaults" (UM-453).
The `getactivecursortime` command gets the time of the active cursor in the Wave window. Returns the time value.

**Syntax**

```
getactivecursortime
[-window <wname>]
```

**Arguments**

- `-window <wname>`
  Specifies an instance of the Wave window that is not the default. Otherwise, the default Wave window is used. Optional. Use the `view` command (CR-263) to change the default window.

**Examples**

```
getactivecursortime
Returns:
  980 ns
```

**See also**

`left` (CR-164), `right` (CR-208)
getactivemarkertime

The `getactivemarkertime` command gets the time of the active marker in the List window. Returns the time value. If `-delta` is specified, returns time and delta.

Syntax

```
getactivemarkertime
[-window <wname>] [-delta]
```

Arguments

`-window <wname>`
Specifies an instance of the List window that is not the default. Otherwise, the default List window is used. Optional. Use the `view` command (CR-263) to change the default window.

`-delta`  
Returns the delta value. Optional. Default is to return only the time.

Examples

```
getactivemarkertime -delta
```

Returns:

```
980 ns, delta 0
```

See also

`down` (CR-139), `up` (CR-231)
The **help** command displays in the Main window a brief description and syntax for the specified command.

**Syntax**

```
help
   [<command> | <topic>]
```

**Arguments**

- `<command>`
  Specifies the command for which you want help. The entry is case and space sensitive. Optional.

- `<topic>`
  Specifies a topic for which you want help. The entry is case and space sensitive. Optional. Specify one of the following six topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commands</td>
<td>Lists all available commands and topics</td>
</tr>
<tr>
<td>debugging</td>
<td>Lists debugging commands</td>
</tr>
<tr>
<td>execution</td>
<td>Lists commands that control execution of your simulation.</td>
</tr>
<tr>
<td>Tcl</td>
<td>Lists all available Tcl commands.</td>
</tr>
<tr>
<td>Tk</td>
<td>Lists all available Tk commands.</td>
</tr>
<tr>
<td>incrTCL</td>
<td>Lists all available incrTCL commands</td>
</tr>
</tbody>
</table>
The `history` command lists the commands you have executed during the current session. History is a Tcl command. For more information, consult the Tcl Man Pages.

**Syntax**

```tcl
history [clear] [keep <value>]
```

**Arguments**

- `clear`
  - Clears the history buffer. Optional.

- `keep <value>`
  - Specifies the number of executed commands to keep in the history buffer. Optional. The default is 50.
lecho

The **lecho** command takes one or more Tcl lists as arguments and pretty-prints them to the Main window. Returns nothing.

**Syntax**

```
lecho
<args> ...
```

**Arguments**

```
<args> ...
Any Tcl list created by a command or user procedure.
```

**Examples**

```
lecho [configure wave]
Prints the Wave window configuration list to the Main window.
```
left

The `left` command searches left (previous) for signal transitions or values in the specified Wave window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a waveform takes on a particular value, or an expression of multiple signals evaluates to true. See the `right` command (CR-208) for related functionality.

The procedure for using left entails three steps: click on the desired waveform; click on the desired starting location; issue the `left` command. (The `seetime` command (CR-216) can initially position the cursor from the command line, if desired.)

Returns: `<number_found> <new_time> <new_delta>`

Syntax

```
left [-expr {<expression>}] [-falling] [-noglitch] [-rising]
[-value <sig_value>] [-window <wname>] [><n]]
```

Arguments

- `expr {<expression>}`
  The waveform display will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window. See "GUI_expression_format" (CR-18) for the format of the expression. The expression must be placed within curly braces.

- `falling`
  Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

- `noglitch`
  Looks at signal values only on the last delta of a time step. For use with `-value` option only. Optional.

- `rising`
  Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

- `value <sig_value>`
  Specifies a value of the signal to match. Must be specified in the same radix in which the selected waveform is displayed. Case is ignored, but otherwise the value must be an exact string match — don't-care bits are not yet implemented. Only one signal may be selected, but that signal may be an array. Optional.

- `window <wname>`
  Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the `view` command (CR-263) to change the default window.
<n>
Specifies to find the nth match. If less than n are found, the number found is returned with
a warning message, and the cursor is positioned at the last match. Optional. The default
is 1.

Examples

left -noglitch -value FF23 2
Finds the second time to the left at which the selected vector transitions to FF23, ignoring
glitches.

left
Goes to the previous transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes,
paths, array constants, and time variables. Such expressions follow the
“GUI_expression_format” (CR-18) and can be built with the aid of the "The GUI Expression
Builder" (UM-305).

left -expr {clk’rising && (mystate == reading) && (/top/u3/addr ==
32’habcd1234)}
Searches left for an expression that evaluates to a boolean 1 when signal
clk just changed
from low to high and signal mystate is the enumeration reading and signal /top/u3/addr
is equal to the specified 32-bit hex constant; otherwise is 0.

left -expr {(/top/u3/addr and 32’hff000000) == 32’hac000000}
Searches left for an expression that evaluates to a boolean 1 when the upper 8 bits of the
32-bit signal /top/u3/adder equals hex ac.

left -expr {((NOW > 23 us) && (NOW < 54 us)) && clk’rising && (mode ==
writing)}
Searches left for an expression that evaluates to a boolean 1 when logfile time is between
23 and 54 microseconds, and clock just changed from low to high and signal mode is
enumeration writing.

▶ Note: Wave window mouse and keyboard shortcuts (UM-274) are also available for next
and previous edge searches. Tab searches right (next) and shift-tab searches left
(previous).

See also

“GUI_expression_format” (CR-18), right (CR-208), seetime (CR-216), view (CR-263)
log

The log command creates a wave log format (WLF) file containing simulation data for all HDL items whose names match the provided specifications. Items (VHDL signals and variables, and Verilog nets and registers) that are displayed using the add list (CR-48) and add wave (CR-57) commands are automatically recorded in the WLF file. The log is stored in a WLF file (formerly a WAV file) in the working directory. By default the file is named vsim.wlf. You can change the default name using the -wlf option of the vsim (CR-298) command.

If no port mode is specified, the WLF file contains data for all items in the selected region whose names match the item name specification.

The WLF file is the source of data for the List and Wave windows. An item that has been logged and is subsequently added to the List or Wave window will have its complete history back to the start of logging available for listing and waving.

Limitations: Verilog memories and VHDL variables can be logged using the variable’s full name only (no wildcards).

Syntax

```
[-recursive] <item_name> ...
```

Arguments

- **-flush**
  Adds region data to the WLF file after each individual log command. Optional. Default is to add region data to the log file only when a command that advances simulation time is executed (e.g., run, step, etc.) or when you quit the simulation.

- **-howmany**
  Returns an integer indicating the number of signals found. Optional.

- **-in**
  Specifies that the WLF file is to include data for ports of mode IN whose names match the specification. Optional.

- **-inout**
  Specifies that the WLF file is to include data for ports of mode INOUT whose names match the specification. Optional.

- **-internal**
  Specifies that the WLF file is to include data for internal items whose names match the specification. Optional.

- **-out**
  Specifies that the WLF file is to include data for ports of mode OUT whose names match the specification. Optional.

- **-ports**
  Specifies that the scope of the search is to include all ports. Optional.
-recursive
   Specifies that the scope of the search is to descend recursively into subregions. Optional; if omitted, the search is limited to the selected region.

=item_name>
   Specifies the item name which you want to log. Required. Multiple item names may be specified. Wildcard characters are allowed. (Note that the WildcardFilter Tcl preference variable identifies types to ignore when matching items with wildcard patterns.)

Examples

log -r /*
   Logs all items in the design.

log -out *
   Logs all output ports in the current design unit.

See also

add list (CR-48), add wave (CR-57), nolog (CR-174), and "Wildcard characters" (CR-16)

► Note: The log command is also known as the "add log" command.
Ishift

The `lshift` command takes a Tcl list as an argument and shifts it in-place, one place to the left, eliminating the 0th element. The number of shift places may also be specified. Returns nothing.

Syntax

```
lshift  <list>  [<amount>]
```

Arguments

- `<list>`
  Specifies the Tcl list to target with `lshift`. Required.
- `<amount>`
  Specifies the number of places to shift. Optional. Default is 1.

Examples

```
proc myfunc args {
    # throws away the first two arguments
    lshift args 2
    ...
}
```

See also

See the Tcl man pages (Help > Tcl Man Pages) for details.
**lsublist**

The `lsublist` command returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern.

**Syntax**

```
lsublist <list> <pattern>
```

**Arguments**

- `<list>`
  Specifies the Tcl list to target with `lsublist`. Required.

- `<pattern>`
  Specifies the pattern to match within the `<list>` using Tcl glob-style matching. Required.

**Examples**

In the example below, variable ‘t’ returns "structure signals source".
```
set window_names "structure signals variables process source wave list dataflow"

set t [lsublist $window_names s*]
```

**See also**

The `set` command is a Tcl command. See the Tcl man pages (Help > Tcl Man Pages) for details.
**macro_option**

This command is available for **UNIX only** (excluding Linux).

The `macro_option` command controls the speed and delay of macro (DO file) playback, plus the level of debugging feedback. If invoked without any options, `macro_option` returns all current settings; returns a specific setting if invoked with an option and no argument; returns the previous setting if invoked with both an option and an argument.

**Syntax**

```
macro_option
    [speed fast | demo] | [delay <delay_time>] | [debug <level>]
```

**Arguments**

- **speed fast | demo**
  
  Set the macro playback speed to fast or demo. Optional.

- **delay <delay_time>**
  
  Set the delay time in milliseconds; delay is the time between events in demo mode. Optional.

- **debug <level>**
  
  Set the debug level from 1 to 9; 9 giving the most feedback. Optional.

**See also**

- `play` (CR-183), `run` (CR-210)
modelsim

The modelsim command starts the ModelSim GUI without prompting you to load a design. This command is valid only for Windows platforms, and may be invoked in one of three ways:

from the DOS prompt
from a ModelSim shortcut
from the Windows Start > Run menu

To use modelsim arguments with a shortcut, add them to the target line of the shortcut’s properties. (Arguments work on the DOS command line too, of course.)

The simulator may be invoked from either the MODELSIM prompt after the GUI starts or from a DO file called by modelsim.

Syntax

```modelsim [-do <macrofile>] [-project <project file>]```

Arguments

- `-do <macrofile>`
  Specifies the DO file to execute when modelsim is invoked. Optional.

  Note: In addition to the macro called by this argument, if a DO file is specified by the STARTUP variable in modelsim.ini, it will be called when the vsim command (CR-298) is invoked.

- `-project <project file>`
  Specifies the modelsim.ini file to load for this session. Optional.

See also

vsim (CR-298), do (CR-138), and "Using a startup file" (UM-452)
next

The next command continues a search after you have invoked the search command. See "search" (CR-212) for more information.

Syntax

```plaintext
next
<win_type> [-window <wname>]
```

Arguments

- `<win_type>`
  Specifies structure, signals, process, variables, wave, list, source, or a unique abbreviation thereof. Required.

- `-window <wname>`
  Specifies an instance of the window that is not the default. Optional. Otherwise, the default window is used. Use the view command (CR-263) to change the default window.
**noforce**

The **noforce** command removes the effect of any active **force** (CR-156) commands on the selected HDL items. The **noforce** command also causes the item’s value to be re-evaluated.

### Syntax

```plaintext
noforce <item_name>  ...
```

### Arguments

- `<item_name>`
  - Specifies the name of a item. Required. Must match an item name used in a previous **force** command (CR-156). Multiple item names may be specified. Wildcard characters are allowed.

### See also

- **force** (CR-156) and "Wildcard characters" (CR-16)
nolog

The nolog command suspends writing of data to the wave log format (WLF) file for the specified signals. A flag is written into the WLF file for each signal turned off, and the GUI displays "No Data-" for the signal(s) until logging (for the signal(s)) is turned back on.

Logging can be turned back on by issuing another log command (CR-166) or by doing a nolog -reset.

Because use of the nolog command adds new information to the WLF file, WLF files created when using the nolog command cannot be read by older versions of the simulator. If you are using dumplog64.c, you will need to get an updated version.

Syntax

nolog
  [-internal]  |  [-howmany]  |  <item_name>  ...  

Arguments

-all
  Turns off logging for all signals currently logged. Optional. Must be used alone without other arguments.

-reset
  Turns logging back on for all signals unlogged. Optional. Must be used alone without other arguments.

-recursive
  Specifies that the scope of the search is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region.

-in
  Specifies that the WLF file is to turn off logging for ports of mode IN whose names match the specification. Optional.

-out
  Specifies that the WLF file is to turn off logging for ports of mode OUT whose names match the specification. Optional.

-inout
  Specifies that the WLF file is to turn off logging for ports of mode INOUT whose names match the specification. Optional.

-ports
  Specifies that the scope of the search is to turn off logging for all ports. Optional.

-internal
  Specifies that the WLF file is to turn off logging for internal items whose names match the specification. Optional.

-howmany
  Returns an integer indicating the number of signals found. Optional.
<item_name>
    Specifies the item name which you want to unlog. Required. Multiple item names may
    be specified. Wildcard characters are allowed.

Examples

    nolog -r /*
    Unlogs all items in the design.

    nolog -out *
    Unlogs all output ports in the current design unit.

See also

    add list (CR-48), add wave (CR-57), log (CR-166)
notepad

The **notepad** command opens a simple text editor. It may be used to view and edit ASCII files or create new files. When a file is specified on the command line, the editor will initially come up in read-only mode. This mode can be changed from the Notepad Edit menu. See "Mouse and keyboard shortcuts" (UM-183) for a list of editing shortcuts.

Returns nothing.

**Syntax**

```
notepad
   [<filename>] [-r | -edit]
```

**Arguments**

- `<filename>`
  Name of the file to be displayed. Optional.

- `-r | -edit`
  Selects the notepad editing mode: -r for read-only, and -edit for edit mode. Optional. Read-only is the default.
The `noview` command closes a window in the ModelSim GUI. To open a window, use the `view` command.

**Syntax**

```
 noview
[*] <window_name>...
```

**Arguments**

- `[*]` (optional)
  - Wildcards can be used, for example: `l*` (List window), `s*` (Signal, Source, and Structure windows), even `*` alone (all windows).
- `<window_name>...` (required)
  - Specifies the ModelSim window type to close. Multiple window types may be used; at least one type (or wildcard) is required. Available window types are:
    - `dataflow`, `list`, `process`, `signals`, `source`, `structure`, `variables`, and `wave`

**Examples**

- `noview wave1`
  - Closes the Wave window named "wave1".
- `noview l*`
  - Closes all List windows.
- `noview s*`
  - Closes all Structure, Signals, and Source windows.

See also

- `view (CR-263)`
nowhen

The `nowhen` command deactivates selected `when` (CR-314) commands.

Syntax

```
nowhen [ <label> ]
```

Arguments

`<label>`
Specifies an individual when command. Optional. Wildcards may be used to select more than one when command.

Examples

```
when -label 99 b {echo "b changed"}
...
nowhen 99
  This nowhen command deactivates the when (CR-314) command labeled 99.

nowhen *
  This nowhen command deactivates all when (CR-314) commands.
```
onbreak

CR-179

onbreak
The onbreak command is used within a macro. It specifies one or more commands to be
executed when running a macro that encounters a breakpoint in the source code. Using the
onbreak command without arguments will return the current onbreak command string.
Use an empty string to change the onbreak command back to its default behavior (i.e.,
onbreak ""). In that case, the macro will be interrupted after a breakpoint occurs (after any
associated bp command (CR-68) string is executed).
onbreak commands can contain macro calls.

Syntax
onbreak
{[<command> [; <command>] ...]}

Arguments
<command>

Any command can be used as an argument to onbreak. If you want to use more than one
command, use a semicolon to separate the commands, or place them on multiple lines.
The entire command string must be placed in curly braces. It is an error to execute any
commands within an onbreak command string following a run (CR-210), run -continue,
or step (CR-222) command. This restriction applies to any macros or Tcl procedures used
in the onbreak command string. Optional.

Examples
onbreak {exa data ; cont}

Examine the value of the HDL item data when a breakpoint is encountered. Then
continue the run command (CR-210).
onbreak {resume}

Resume execution of the macro file on encountering a breakpoint.

See also
abort (CR-44), bd (CR-63), bp (CR-68), do (CR-138), onerror (CR-181), resume (CR-207),
status (CR-221)

ModelSim SE Command Reference


onElabError

The onElabError command specifies one or more commands to be executed when an error is encountered during elaboration. The command is used by placing it within the modelsim.tcl file or a macro. During initial design load onElabError may be invoked from within the modelsim.tcl file; during a simulation restart onElabError may be invoked from a macro.

Use the onElabError command without arguments to return to a prompt.

Syntax

onElabError
   { [ <command> ; <command> ] ... }

Arguments

<command>
   Any command can be used as an argument to onElabError. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. Optional.

See also

do (CR-138)
onerror

The onerror command is used within a macro; it specifies one or more commands to be executed when a running macro encounters an error. Using the onerror command without arguments will return the current onerror command string. Use an empty string to change the onerror command back to its default behavior (i.e., onerror "").. Use onerror with a resume command (CR-207) to allow an error message to be printed without halting the execution of the macro file.

Syntax

\[
\text{onerror} \\
\{ [<\text{command}> ; <\text{command}>] \ldots \}
\]

Arguments

<command>

Any command can be used as an argument to onerror. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. Optional.

Example

\[
\text{onerror} \{ \text{quit -f} \}
\]

Forces the simulator to quit if an error is encountered while the macro is running.

See also

abort (CR-44), do (CR-138), onbreak (CR-179), resume (CR-207), status (CR-221)

Note: You can also set the global OnErrorDefaultAction Tcl variable in the pref.tcl file to dictate what action ModelSim takes when an error occurs. The onerror command is invoked only when an error occurs in the macro file that contains the onerror command. Conversely, OnErrorDefaultAction will run even if the macro does not contain a local onerror command. This can be useful when you run a series of macros from one script, and you want the same behavior across all macros.
pause

The **pause** command placed within a macro interrupts the execution of that macro.

**Syntax**

```
pause
```

**Arguments**

None.

**Description**

When you execute a macro and that macro gets interrupted, the prompt will change to:

```
VSIM (pause)?>
```

This “pause” prompt reminds you that a macro has been interrupted.

When a macro is paused, you may invoke another macro, and if that one gets interrupted, you may even invoke another — up to a nesting level of 50 macros.

If the status of nested macros gets confusing, use the **status** command (CR-221). It will show you which macros are interrupted, at what line number, and show you the interrupted command.

To resume the execution of the macro, use the **resume** command (CR-207). To abort the execution of a macro use the **abort** command (CR-44).

**See also**

- **abort** (CR-44), **do** (CR-138), **resume** (CR-207), **run** (CR-210)
play

This command is available for UNIX only (excluding Linux).
The play command replays a sequence of keyboard and mouse actions that were previously saved to a file with the record command (CR-201). Returns nothing.

Note: Play returns immediately; the playback proceeds in the background. Caution must be used when putting play commands in do (macro) files.

Syntax

play
   <filename>

Arguments

<filename>
   Specifies the recorded file to replay. Required.

Playback controls

The following Tcl set commands control the playback type and speed by setting the play_macro() global variables. The commands are invoked from the ModelSim command line.

set play_macro(speed)
   Specify the playback speed: either demo (with the delay specified below), or fast (no delays).

set play_macro(delay)
   Specifies the delay time in milliseconds. Controls the speed of playback in demo mode.

See also

macro_option (CR-170), record (CR-201)
power add

The **power add** command is used prior to the **power report** command (CR-185). Data produced by these commands can be translated (by a Synopsys utility) to drive the Synopsys power analysis tools. This command specifies the signals or nets to track for power information. Returns nothing.

**Syntax**

```
power add
   [-in] [-inout] [-internal] [-out] [-ports] [-r] <signalsOrNets> ...
```

**Arguments**

- `-in` Specifies only inputs. Optional.
- `-inout` Specifies only inouts. Optional.
- `-internal` Specifies only design internal signals or nets. Optional.
- `-out` Specifies only outputs. Optional.
- `-ports` Specifies only design ports. Optional.
- `-r` Searches recursively on a wildcard specified for the signal or net. Optional.
- `<signalsOrNets> ...` Specifies the signal or net to track. Required. Multiple names or wildcards may be used. Must refer to VHDL signals of type bit, std_logic, or std_logic_vector, or to Verilog nets.

When using wildcards, the -in, -inout, -internal, -out, and -ports arguments filter the qualifying signals. If you specify more than one of these arguments, the logical OR of the arguments is performed.

**See also**

- **power report** (CR-185), **power reset** (CR-186)

See the Synopsys Power documentation for more information.
power report

The **power report** command is used subsequent to the **power add** command (CR-184). Data produced by these commands can be translated (by a Synopsys utility) to drive the Synopsys power analysis tools. This command writes out the power information for the specified signals or nets. The report can be written to a file or to the Main window. Returns nothing.

**Syntax**

```plaintext
power report
  [-all] [-noheader] [-file <filename>]
```

**Arguments**

- **-all**
  - Writes information on all items logged. Optional.
- **-noheader**
  - Suppresses the header to aid in post processing. Optional.
- **-file <filename>**
  - Specifies a filename for the power report. Optional. Default is to write the report to the Main window.

**Description**

The report format for each line is:

```
signal path, toggle count, hazard count, time at a 1, time at a 0, time at an X
```

- toggle count is the number of 0->1 and 1->0 transitions
- hazard count is the number of 0/1->X, and X->0/1 transitions

  Note that if a signal is initialized at X, and later transitions to 0 or 1, it is not counted as a hazard.

- times are the times spent at each of the three respective states

You will also need to know the total simulation time.

**See also**

- **power add** (CR-184), **power reset** (CR-186)

See the Synopsys Power documentation for more information.
power reset

The **power reset** command selectively resets power information to zero for the signals or nets specified with the **power add** command (CR-184). Returns nothing.

**Syntax**

```
power reset
   <signalsOrNets> ...
```

**Arguments**

- **-all**
  - Resets all signals/nets. Optional.
- **-in**
  - Resets only inputs. Optional.
- **-inout**
  - Resets only inouts. Optional.
- **-out**
  - Resets only outputs. Optional.
- **-internal**
  - Resets only design internal signals or nets. Optional.
- **-ports**
  - Resets only design ports. Optional.
- **-r**
  - Searches recursively on a wildcard specified for the signal or net. Optional.

<signalsOrNets> ...

Specifies the signal or net to reset. Required. Multiple names or wildcards may be used.

**See also**

- **power add** (CR-184), **power report** (CR-185)

See the Synopsys Power documentation for more information.
printenv

The `printenv` command echoes to the Main window the current names and values of all environment variables. If variable names are given as arguments, prints only the names and values of the specified variables. Returns nothing. All results go to the Main window.

Syntax

```
printenv
   [<var>...]  
```

Arguments

`
<var>...
`
Specifies the name(s) of the environment variable(s) to print. Optional.

Examples

```
printenv
   Prints all environment variable names and their current values. For example,
   # CC = gcc
   # DISPLAY = srl:0.0
   ...

printenv USER HOME
   Prints the specified environment variables:
   # USER = vince
   # HOME = /scratch/srl/vince
```
profile clear

The `profile clear` command is used to clear any data that has been gathered during previous `run` commands. After this command is executed, all profiling data will be reset.

This command has no effect on the current profiling session. The last `profile on` or `profile off` command will still be in effect.

Syntax

```
profile clear
```

Arguments

None

See also

- `profile interval` (CR-189)
- `profile off` (CR-190)
- `profile on` (CR-191)
- `profile option` (CR-192)
- `profile report` (CR-193)

▶ **Note:** Profiling must be active when this command is invoked. Use the `profile on` command (CR-191) to begin profiling.
profile interval

The profile interval command allows you to select the frequency with which the profiler collects samples during a run command.

Syntax

```
profile interval
  [<sample_frequency>]
```

Arguments

```
<sample_frequency>
```
An integer value from 1 to 999 that represents how many milliseconds to wait between each sample collected during a profiled simulation run. Default is 10 ms.

If the sample-frequency is not supplied, the profile interval command returns the current sample frequency.

See also

- profile clear (CR-188), profile off (CR-190), profile on (CR-191), profile option (CR-192), profile report (CR-193)

▶ Note: Profiling must be active when this command is invoked. Use the profile on command (CR-191) to begin profiling.
profile off

The `profile off` command is used to discontinue runtime profiling.

Syntax

```
profile off
```

Arguments

None

See also

profile on

The **profile on** command is used to enable runtime analysis of where your simulation is spending its time. After this command is executed, every subsequent `run` command will be profiled.

**Syntax**

```
profile on
```

**Arguments**

None

**See also**

- `profile clear` (CR-188)
- `profile interval` (CR-189)
- `profile off` (CR-190)
- `profile option` (CR-192)
- `profile report` (CR-193)
profile option

The **profile option** command changes how profiling data are reported.

Syntax

```
profile option
  collapse_sections | raw_data
```

Arguments

**collapse_sections**

Groups profiling data by section. A section consists of regions of code such as VHDL processes, functions or Verilog `always` blocks. By default all profiling data is reported on a per line basis. Required if **raw_data** isn’t specified.

▶ **Note:** This option must be specified before the **run** command (CR-210) is executed.

**raw_data**

Reports the raw number of samples that occurred in a line or a section. By default all profiling results are reported on a percentage basis. Required if **collapse_sections** isn’t specified.

See also

- **profile clear** (CR-188), **profile interval** (CR-189), **profile off** (CR-190), **profile on** (CR-191), **profile report** (CR-193)

▶ **Note:** Profiling must be active when this command is invoked. Use the **profile on** command (CR-191) to begin profiling.
The **profile report** command is used to produce a textual output of the profiling statistics that have been gathered up to this point. (Selecting **Tools > Profile > View hierarchical profile** (Main window) and **Tools > Profile > View ranked profile** allows you to view this data more interactively.)

### Syntax

```
profile report
   [-hierarchical | -ranked] [-file <filename>] [-cutoff <percentage>]
```

### Arguments

- **-hierarchical**
  Report a hierarchical listing (Default). Optional.

- **-ranked**
  Report a ranked listing. Optional.

- **-file <filename>**
  Specifies a file name for the report. Optional. Default is to write the report to the Main window.

- **-cutoff <percentage>**
  Filter entries in the report that had less than <percentage> of time spent in them. Optional. Default is to report all entries (i.e. 0%).

### See also

- **profile clear** (CR-188), **profile interval** (CR-189), **profile off** (CR-190), **profile on** (CR-191), **profile option** (CR-192)

▶ **Note:** Profiling must be active when this command is invoked. Use the **profile on** command (CR-191) to begin profiling.
The `project` commands are used to perform common operations on projects. Use this command outside of a simulation session.

**Syntax**

```
project
  [addfile <filename>] | [close] | [compileall] | [delete <project>] | [env]
  | [history] | [new <home_dir> <proj_name> <defaultlibrary>] |
  [open <project>] | [removefile <filename>]
```

**Arguments**

- `addfile <filename>`
  Adds the specified file to the current open project. Optional.

- `close`
  Closes the current project. Optional.

- `compileall`
  Compiles all files in the current project. Optional.

- `delete <project>`
  Deletes a specified project file. Optional.

- `env`
  Returns the current project file. Optional.

- `history`
  Lists a history of manipulated projects. Optional.

- `new <home_dir> <proj_name> <defaultlibrary>`
  Creates a new project under a specified home directory with a specified name and a default library. Optional.

- `open <project>`
  Opens a specified project file, making it the current project. Changes the current working directory to the project’s directory. Optional.

- `removefile <filename>`
  Removes the specified file from the current project. Optional.

**Examples**

```
vsim> project open /user/george/design/test3/test3.mpf
  Makes /user/george/design/test3 the current project and changes the current working directory to /user/george/design/test3.

vsim> project compile all
  Executes current project library build scripts.
```
property list

The property list command changes one or more properties of the specified signal, net or register in the List window (UM-204). The properties correspond to those you can set by selecting View > Signal Properties (List window). At least one argument must be used.

Syntax

property list
  [-window <wname>] [-label <label>] [-radix <radix>]
  [-trigger <setting>] [-width <number>] <pattern>

Arguments

- window <wname>
  Specifies a particular List window when multiple instances of the window exist (e.g., list2). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the view command (CR-263).

- label <label>
  Specifies the label to appear at the top of the List window column. Optional.

- radix <radix>
  The listed value <radix> can be specified as: Symbolic, Bin, Oct, Dec, Hex, or Def. Optional. Def stands for default radix which is set using the radix command (CR-200).

- trigger <setting>
  Valid settings are 0 or 1. Setting trigger to 1 will enable the List window to be triggered by changes in the items matching the specified pattern. Optional.

- width <number>
  Valid numbers are 1 through 256. Specifies the desired column width for the items matching the specified pattern. Optional.

- pattern
  Specifies a name or wildcard pattern to match the full path names of the signals, nets or registers for which you are defining the property change. Required.

To change the time or delta column widths, use these patterns:

TIME or DELTA
property wave

The property wave command changes one or more properties of the specified signal, net or register in the Wave window (UM-246). The properties correspond to those you can set by selecting View > Signal Properties (Wave window). At least one argument must be used.

Syntax

```plaintext
property wave
   [-window <wname>] [-color <color>] [-format <format>] [-height <number>]
   [-offset <number>] [-radix <radix>] [-scale <float>] <pattern>
```

Arguments

- **-window <wname>**
  Specifies a particular Wave window when multiple instances of the window exist (e.g., wave2). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the `view` command (CR-263).

- **-color <color>**
  Specifies the color to be used for the waveform. Optional.

- **-format <format>**
  The waveform <format> can be expressed as:
  - **analog**
    Displays a waveform whose height and position is determined by the -scale and -offset values (shown below). Optional.
  - **literal**
    Displays the waveform as a box containing the item value (if the value fits the space available). Optional.
  - **logic**
    Displays values as 0, 1, X, or Z. Optional.

- **-height <number>**
  Specifies the height (in pixels) of the waveform. Optional.

- **-offset <number>**
  Specifies the waveform position offset in pixels. Valid only when -format is specified as analog. Optional.

- **-radix <radix>**
  The <radix> can be expressed as: Symbolic, Bin, Oct, Dec, Hex, or Def. Choosing symbolic means that item values are not translated. Optional. Def stands for default radix which is set using the `radix` command (CR-200).

- **-scale <float>**
  Specifies the waveform scale relative to the unscaled size value of 1. Valid only when -format is specified as analog. Optional.

- **<pattern>**
  Specifies a name or wildcard pattern to match the full path names of the signals, nets or registers for which you are defining the property change. Required.
pwd

The Tcl `pwd` command displays the current directory path in the Main window.

Syntax

```
pwd
```

Arguments

None.
quietly

The `quietly` command turns off transcript echoing for the specified command.

Syntax

```
quietly
<command>
```

Arguments

```
<command>
```

Specifies the command for which to disable transcript echoing. Required. Any results normally echoed by the specified command will not be written to the Main window transcript. To disable echoing for all commands use the `transcript` command (CR-229) with the `-quietly` option.

See also

`transcript` (CR-229)
quit

The `quit` command exits the simulator.

Syntax

```
quit
```

Arguments

- `-f` or `-force`
  Quits without asking for confirmation. Optional; if this argument is omitted, ModelSim asks you for confirmation before exiting. (The `-f` and `-force` arguments are equivalent.)

- `-sim`
  Unloads the current design in the simulator without exiting ModelSim. All files opened by the simulation will be closed including the WLF file (`vsim.wlf`).

▶ **Note:** If you want to stop the simulation using a `when` command (CR-314), you must use a `stop` command (CR-223) within your `when` statement. DO NOT use an `exit` command (CR-152) or a `quit` command. The `stop` command acts like a breakpoint at the time it is evaluated.
**radix**

The `radix` command specifies the default radix to be used for the current simulation. The command can be used at any time. The specified radix is used for all commands (force (CR-156), examine (CR-149), change (CR-72), etc.) as well as for displayed values in the Signals, Variables, Dataflow, List, and Wave windows. You can change the default radix permanently by editing the DefaultRadix (UM-447) variable in the `modelsim.ini` file.

**Syntax**
```
radix [-symbolic | -binary | -octal | -decimal | -hexadecimal | -unsigned | -ascii]
```

**Arguments**
Entries may be truncated to any length. For example, `-symbolic` could be expressed as `-s` or `-sy`, etc. Optional.

Also, `-signed` may be used as an alias for `-decimal`. The `-unsigned` radix will display as unsigned decimal. The `-ascii` radix will display a Verilog item as a string equivalent using 8 bit character encoding.

If no arguments are used, the command returns the current default radix.
record

This command is available for UNIX only (excluding Linux).
The record command starts recording a replayable trace of all keyboard and mouse actions. Record and play operations may also be run from the macro-helper menu item of the macro menu. Returns nothing.

Syntax

```
record
 [<filename>]
```

Arguments

<filename>
  Specifies the file for the saved recording. If <filename> is not specified, the recording terminates.

See also

macro_option (CR-170), play (CR-183)
**report**

The **report** command displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation.

**Syntax**

```
report
  simulator control | simulator state
```

**Arguments**

- **simulator control**
  Displays the current values for all simulator control variables.

- **simulator state**
  Displays the simulator state variables relevant to the current simulation.

**Examples**

```
report simulator control
  Displays all simulator control variables.
  # UserTimeUnit = ns
  # RunLength = 100
  # IterationLimit = 5000
  # BreakOnAssertion = 3
  # DefaultForceKind = default
  # IgnoreNote = 0
  # IgnoreWarning = 0
  # IgnoreError = 0
  # IgnoreFailure = 0
  # CheckpointCompressMode = 1
  # NumericStdNoWarnings = 0
  # StdArithNoWarnings = 0
  # PathSeparator = /
  # DefaultRadix = symbolic
  # DelayFileOpen = 0

report simulator state
  Displays all simulator state variables. Only the variables that relate to the design being simulated are displayed:
  # now = 0.0
  # delta = 0
  # library = work
  # entity = type_clocks
  # architecture = full
  # resolution = 1ns
```
Viewing preference variables

Preference variables have more to do with the way things look (but not entirely) rather than controlling the simulator. You can view preference variables from the Preferences dialog box. Select the Tools > Edit Preferences.

See also

"Preference variables located in INI files" (UM-444), and "Preference variables located in Tcl files" (UM-454)
**restart**

The **restart** command reloads the design elements and resets the simulation time to zero. Only design elements that have changed are reloaded. (Note that SDF files are always reread during a restart.) Shared libraries are handled as follows during a restart:

- Shared libraries that implement VHDL foreign architectures only are reloaded at each restart when the architecture is elaborated (unless the `-keeploaded` option to the `vsim` command (CR-298) is used).
- Shared libraries loaded from the command line (-foreign and -pli options) and from the Veriuser entry in the `modelsim.ini` file are reloaded.
- Shared libraries that implement VHDL foreign subprograms remain loaded (they are not reloaded) even if they also contain code for a foreign architecture.

To handle restarts with Verilog PLI applications, you need to define a Verilog user-defined task or function, and register a miscf class of callback. See "Verilog PLI/VPI" (UM-121) for more information on the Verilog PLI.

**Syntax**

```
restart
```

**Arguments**

- **-force**  
  Specifies that the simulation will be restarted without requiring confirmation in a popup window. Optional.

- **-nobreakpoint**  
  Specifies that all breakpoints will be removed when the simulation is restarted. Optional. The default is for all breakpoints to be reinstalled after the simulation is restarted.

- **-nolist**  
  Specifies that the current List window environment will not be maintained after the simulation is restarted. Optional. The default is for all currently listed HDL items and their formats to be maintained.

- **-nolog**  
  Specifies that the current logging environment will not be maintained after the simulation is restarted. Optional. The default is for all currently logged items to continue to be logged.

- **-nowave**  
  Specifies that the current Wave window environment will not be maintained after the simulation is restarted. Optional. The default is for all items displayed in the Wave window to remain in the window with the same format.

**Note:** You can configure defaults for the restart command by setting the `DefaultRestartOptions` variable in the `modelsim.ini` file. See "Restart command defaults" (UM-453).
See also

checkpoint (CR-82), restore (CR-206), vsim (CR-298), "How to use checkpoint/restore" (UM-488), "The difference between checkpoint/restore and restarting" (UM-489)
The **restore** command restores the state of a simulation that was saved with a **checkpoint** command (CR-82) during the current invocation of VSIM (called a "warm restore").

The items restored are: simulation kernel state, *vsim.wlf* file, HDL items listed in the List and Wave windows, file pointer positions for files opened under VHDL and under Verilog $fopen, and the saved state of foreign architectures.

If you want to **restore** while running VSIM, use this command. If you want to start up VSIM and restore a previously-saved checkpoint, use the **-restore** switch with the **vsim** command (CR-298) (called a "cold restore").

**Note:** Checkpoint/restore allows a cold restore, followed by simulation activity, followed by a warm restore back to the original cold-restore checkpoint file. Warm restores to checkpoint files that were not created in the current run are not allowed except for this special case of an original cold restore file.

### Syntax

```
restore
    [-nocompress] <filename>
```

### Arguments

- **-nocompress**
  Specifies that the checkpoint file was not compressed when saved. Optional. Compression of the checkpoint file is controlled either by the `CheckpointCompressMode` variable in the `modelsim.ini` file or by the **-nocompress** argument to vsim.

- `<filename>`
  Specifies the name of the checkpoint file. Required.

### See also

- **checkpoint** (CR-82), **vsim** (CR-298), "The difference between checkpoint/restore and restarting" (UM-489)
The `resume` command is used to resume execution of a macro file after a `pause` command (CR-182), or a breakpoint. It may be input manually or placed in an `onbreak` (CR-179) command string. (Placing a `resume` command in a `bp` (CR-68) command string does not have this effect.) The `resume` command can also be used in an `onerror` (CR-181) command string to allow an error message to be printed without halting the execution of the macro file.

**Syntax**

```
resume
```

**Arguments**

None.

**See also**

right

The **right** command searches right (next) for signal transitions or values in the specified Wave window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a waveform takes on a particular value, or an expression of multiple signals evaluates to true. See the **left** command (CR-164) for related functionality.

The procedure for using **right** entails three steps: click on the desired waveform; click on the desired starting location; issue the **right** command. (The **settime** command (CR-216) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

**Syntax**

right

[-expr {<expression>}] [-falling] [-noglitch] [-rising]
[-value <sig_value>] [-window <wname>] [<n>]

**Arguments**

-expr {<expression>}

The waveform display will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window.

See "GUI_expression_format" (CR-18) for the format of the expression. The expression must be placed within curly braces.

-falling

Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

-noglitch

Looks at signal values only on the last delta of a time step. For use with the -value option only. Optional.

-rising

Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

-value <sig_value>

Species a value of the signal to match. Must be specified in the same radix that the selected waveform is displayed. Case is ignored, but otherwise the value must be an exact string match -- don't-care bits are not yet implemented. Only one signal may be selected, but that signal may be an array. Optional.

-window <wname>

Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the **view** command (CR-263) to change the default window.
<n>
Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the cursor is positioned at the last match. Optional. The default is 1.

**Examples**

```
right -noglitch -value FF23 2
Finds the second time to the right at which the selected vector transitions to FF23, ignoring glitches.
```

```
right
Goes to the next transition on the selected signal.
```

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-18) and can be built with the aid of the "The GUI Expression Builder" (UM-305).

```
right -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}
Searches right for an expression that evaluates to a boolean 1 when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /top/u3/addr is equal to the specified 32-bit hex constant; otherwise is 0.
```

```
right -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
Searches right for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal /top/u3/addr equals hex ac.
```

```
right -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)}
Searches right for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal mode is enumeration writing.
```

▲ **Note:** Wave window mouse and keyboard shortcuts (UM-274) are also available for next and previous edge searches. Tab searches right (next) and shift-tab searches left (previous).

**See also**

"GUI_expression_format" (CR-18), **left** (CR-164), **seetime** (CR-216), **view** (CR-263)
run

The **run** command advances the simulation by the specified number of timesteps.

Syntax

```plaintext
run
    [<timesteps>[<time_units>]] | -all | -continue | -next | -step | -stepover
```

Arguments

- `<timesteps>[<time_units>]`
  - Specifies the number of timesteps for the simulation to run. The number may be fractional, or may be specified absolute by preceding the value with the character @. Optional. In addition, optional `<time_units>` may be specified as:

    - **fs**, **ps**, **ns**, **us**, **ms**, or **sec**

  - The default `<timesteps>` and `<time_units>` specifications can be changed during a ModelSim session by selecting **Simulate > Simulation Options** (Main window). See "Setting default simulation options" (UM-297). Time steps and time units may also be set with the **RunLength** (UM-449) and **UserTimeUnit** (UM-450) variables in the `modelsim.ini` file.

- `-all`
  - Causes the simulator to run the current simulation forever, or until it hits a breakpoint or specified break event. Optional.

- `-continue`
  - Continues the last simulation run after a **step** (CR-222) command, **step -over** command or a breakpoint. A **run -continue** command may be input manually or used as the last command in a **bp** (CR-68) command string. Optional.

- `-next`
  - Causes the simulator to run to the next event time. Optional.

- `-step`
  - Steps the simulator to the next HDL statement. Optional.

- `-stepover`
  - Specifies that VHDL procedures, functions and Verilog tasks are to be executed but treated as simple statements instead of entered and traced line by line. Optional.
Examples

run 1000
   Advances the simulator 1000 timesteps.

run 10.4 ms
   Advances the simulator the appropriate number of timesteps corresponding to 10.4 milliseconds.

run @8000
   Advances the simulator to timestep 8000.

See also

step (CR-222)
search

The `search` command searches the specified window for one or more items matching the specified pattern(s). The search starts at the item currently selected, if any; otherwise it starts at the window top. The default action is to search downward until the first match, then move the selection to the item found, and return the index of the item found. The search can be continued using the `next` command.

Returns the index of a single match, or a list of matching indices. Returns nothing if no matches are found.

Syntax

```
search
  <win_type> [-window <wname>] [-all] [-field <n>] [-toggle]
  [-count <n>] <pattern>
```

Arguments for all windows

- `<win_type>`
  Specifies structure, signals, process, variables, wave, list, source, or a unique abbreviation thereof. Required.

- `-window <wname>`
  Specifies an instance of the window that is not the default. Optional. Otherwise, the default window is used. Use the `view` command (CR-263) to change the default window.

- `-forward`
  Search in the forward direction. Optional. This is the default.

- `-backward`
  Search in the reverse direction. Optional. Default is forward.

- `<pattern>`
  String or glob-style wildcard pattern. Required. Must be the last argument specified.

Arguments, for all EXCEPT the Source window

- `-all`
  Finds all matches and returns a list of the indices of all items that match. Optional.

- `-field <n>`
  Selects different fields to test, depending on the window type:

```
<table>
<thead>
<tr>
<th>Window</th>
<th>n=1</th>
<th>n=2</th>
<th>n=3</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>instance</td>
<td>entity/module</td>
<td>architecture</td>
<td>instance</td>
</tr>
<tr>
<td>signals</td>
<td>name</td>
<td>-</td>
<td>cur. value</td>
<td>name</td>
</tr>
<tr>
<td>process</td>
<td>status</td>
<td>process label</td>
<td>fullpath</td>
<td>fullpath</td>
</tr>
</tbody>
</table>
```
Default behavior for the List window is to attempt to match the label and if that fails, try to match the full signal name.

-`toggle`
  Adds signals found to the selection. Does not do an initial clear selection. Optional. Otherwise deselects all and selects only one item.

-`wrap`
  Specifies that the search continue from the top of the window after reaching the bottom. Optional. This is the default.

-`nowrap`
  Specifies that the search stop at the bottom of the window and not continue searching at the top. Optional. The default is to wrap.

Arguments, Source window only

-`exact`
  Search for an exact match. Optional.

-`regexp`
  Use the pattern as a Tcl regular expression. Optional.

-`nocase`
  Ignore case. Optional. Default is to use case.

-`count <n>`
  Search for the nth match. Optional. Default is to search for the first match.

Description

With the `-all` option, the entire window is searched, the last item matching the pattern is selected, and a Tcl list of all corresponding indices is returned.

With the `-toggle` option, items found are selected in addition to the current selection.

For the List window, the search is done on the names of the items listed, that is, across the header. To search for values of signals in the List window, use the `down` command (CR-139) and `up` command (CR-231). Likewise, in the Wave window, the search is done on signal names. To search for signal values in the Wave window, use the `right` command (CR-208) and the `left` command (CR-164). You can also select `Edit > Search` in both windows.

See also

`find` (CR-153), `next` (CR-172), `view` (CR-263)
searchlog

The searchlog command searches one or more of the currently open logfiles for a specified condition. It can be used to search for rising or falling edges, for signals equal to a specified value, or for when a generalized expression becomes true.

Syntax

searchlog
[-count <n>] [-deltas] [-env <path>] [-expr {<expr>}] [-reverse]
[-rising | -falling | -anyedge] [-startDelta <num>] <startTime>
[-value <string>] <pattern>

If at least one match is found, it returns the time (and optionally delta) at which the last match occurred and the number of matches found, in a Tcl list:

{{<time>} <matchCount>}

where <time> is in the format <number> <unit>. If the -deltas option is specified, the delta of the last match is also returned:

{{<time>} <delta> <matchCount>}

If no matches are found, a TCL_ERROR is returned. If one or more matches are found, but less than the number requested, it is not considered an error condition, and the time of the farthest match is returned, with the count of the matches found.

Arguments

- count <n>
  Specifies to search for the <n>-th occurrence of the match condition, where <n> is a positive integer. Optional.

- deltas
  Indicates to test for match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step. Optional.

- env <path>
  Provides a design region in which to look for the signal names. Optional.

- expr {<expr>}
  Specifies a general expression of signal values and simulation time. Optional. searchlog will search until the expression evaluates to true. The expression must have a boolean result type. See "GUI_expression_format" (CR-18) for the format of the expression.

- reverse
  Specifies to search backwards in time from <startTime>. Optional.

- rising | -falling | -anyedge
  Specifies an edge to look for on a scalar signal. Optional. This option is ignored for compound signals. If no options are specified, the default is -anyedge.

- startDelta <num>
  Indicates a simulation delta cycle on which to start. Optional.
<startTime>
Specifies the simulation time at which to start the search. Required. The time may be specified as an integer number of simulation units, or as \(<num> <timeUnit>\), where \(<num>\) can be integer or with a decimal point, and \(<timeUnit>\) is one of the standard VHDL time units (fs, ps, ns, us, ms, sec).

-value <string>
Specifies to search until a single scalar or compound signal takes on this value. Optional.

<pattern>
Specifies one or more signal names or wildcard patterns of signal names to search on. Required (unless the -expr option is used).

See also

virtual signal (CR-282), virtual log (CR-274), virtual nolog (CR-277)
seetime

The `seetime` command scrolls the List or Wave window to make the specified time visible. For the List window, a delta can be optionally specified as well.

Returns nothing

**Syntax**

```
seetime
   list|wave [-window <wname>] [-select] [-delta <num>] <time>
```

**Arguments**

- `list|wave` Specifies the target window type. Required.
- `[-window <wname>]` Specifies an instance of the Wave or List window that is not the default. Optional. Otherwise, the default Wave or List window is used. Use the `view` command (CR-263) to change the default window.
- `[-select]` Also move the active cursor or marker to the specified time (and optionally, delta). Optional. Otherwise, the window is only scrolled.
- `[-delta <num>]` For the List window when deltas are not collapsed, this option specifies a delta. Optional. Otherwise, delta 0 is selected.
- `<time>` Specifies the time to be made visible. Required.
shift

The `shift` command shifts macro parameter values left one place, so that the value of parameter $2$ is assigned to parameter $1$, the value of parameter $3$ is assigned to $2$, etc. The previous value of $1$ is discarded.

The `shift` command and macro parameters are used in macro files. If a macro file requires more than nine parameters, they can be accessed using the `shift` command.

To determine the current number of macro parameters, use the `argc` (UM-456) variable.

Syntax

```
shift
```

Arguments

None.

Description

For a macro file containing nine macro parameters defined as $1$ to $9$, one `shift` command shifts all parameter values one place to the left. If more than nine parameters are named, the value of the tenth parameter becomes the value of $9$ and can be accessed from within the macro file.

See also

`do` (CR-138)
show

The `show` command lists HDL items and subregions visible from the current environment. The items listed include:

- **VHDL**
  - signals and instances
- **Verilog**
  - nets, registers, tasks, functions, instances and memories

The `show` command returns its results as a formatted Tcl string; to eliminate formatting, use the `Show` command.

**Syntax**

```
show [-all] [<pathname>]
```

**Arguments**

- `-all`
  - Display all names at and below the specified path recursively. Optional.
- `<pathname>`
  - Specifies the pathname of the environment for which you want the items and subregions to be listed. Optional; if omitted, the current environment is assumed.

**Examples**

```
show
    List the names of all the items and subregion environments visible in the current environment.
show /uut
    List the names of all the items and subregions visible in the environment named /uut.
show sub_region
    List the names of all the items and subregions visible in the environment named sub_region which is directly visible in the current environment.
```

**See also**

- `environment` (CR-148), `find` (CR-153)
The `simstats` command returns performance-related statistics about the simulation.

If executed without arguments, the command returns a list of pairs like the following:

```
{memory 57376} {{working set} 56152} {time 0} {{cpu time} 0} {context 0} /
{{page faults} 0}
```

See the arguments below for descriptions of each pair.

► **Note:** Some of the values may not be available on all platforms and other values may be approximates. Different operating systems report these numbers differently.

**Syntax**

```
simstats
[ memory | working | time | cpu | context | faults]
```

**Arguments**

- **memory**
  Returns the amount of virtual memory that the OS has allocated for vsim. Optional.

- **working**
  Returns the portion of allocated virtual memory that is currently being used by all vsim processes. Optional. If this number exceeds memory size, you will encounter performance degradation.

- **time**
  Returns the cumulative "wall clock time" of the run commands. Optional.

- **cpu**
  Returns the cumulative processor time of the run commands. Optional. Processor time differs from wall clock time in that processor time is only counted when the cpu is actually running vsim. If vsim is swapped out for another process, cpu time does not increase.

- **context**
  Returns the number of context swaps (vsim being swapped out for another process) that have occurred during the run commands. Optional.

- **faults**
  Returns the number of page faults that have occurred during the run commands. Optional.
splitio

The `splitio` command operates on a VHDL inout or out port to create a new signal having the same name as the port suffixed with "__o". The new signal mirrors the output driving contribution of the port.

Syntax

```
splitio [-outalso | -outonly] [-r] <signal_name>...
```

Arguments

- `-outalso`
  Allows `splitio` to work on out ports as well as inout ports. Optional.

- `-outonly`
  Allows `splitio` to work only on out ports. Optional.

- `-r`
  Specifies that the port selection occurs recursively into subregions. Optional; if omitted, included ports are limited to the current region.

- `<signal_name>...`
  Specifies the VHDL port. Operates only on inout ports by default; out ports may be specified with the options above. Separate multiple port names with spaces. Required. Wildcards can be used.

Examples

The `splitio` command operates on inout or out ports and silently ignores any other signals specified. The new signals created may be specified in any `vsmi` (CR-298) commands that operate on signals. These signals appear to be out ports to the signal selection options on `vsmi` commands. For example,

```
splitio /data
```

creates signal data__o if data is an inout port.
status

The **status** command lists all currently interrupted macros. The listing shows the name of each interrupted macro, the line number at which it was interrupted, and prints the command itself. It also displays any `onbreak` (CR-179) or `onerror` (CR-181) commands that have been defined for each interrupted macro.

**Syntax**

```plaintext
status
```

**Arguments**

None.

**Examples**

The transcript below contains examples of `resume` (CR-207), and `status` commands.

```plaintext
VSIM (pause) 4> status
# Macro resume_test.do at line 3 (Current macro)
# command executing: "pause"
# is Interrupted
# ONBREAK commands: "resume"
# Macro startup.do at line 34
# command executing: "run 1000"
# is Interrupted
# ONBREAK commands: "resume"
VSIM (pause) 5> resume
# Resuming execution of macro resume_test.do at line 4
```

**See also**

`abort` (CR-44), `do` (CR-138), `pause` (CR-182), `resume` (CR-207)
**step**

The `step` command steps to the next HDL statement. Current values of local variables may be observed at this time using the Variables window. VHDL procedures and functions and Verilog tasks and functions can optionally be skipped over. When a wait statement or end of process is encountered, time advances to the next scheduled activity. The Process and Source windows will then be updated to reflect the next activity.

**Syntax**

```plaintext
step [-over] [<n>]
```

**Arguments**

- `-over`
  Specifies that VHDL procedures and functions and Verilog tasks and functions should be executed but treated as simple statements instead of entered and traced line by line. Optional.

- `<n>`
  Any integer. Optional. Will execute ‘n’ steps before returning.

**See also**

`run` (CR-210)
The **stop** command is used with the **when** command (CR-314) to stop simulation in batch files. The **stop** command has the same effect as hitting a breakpoint. The **stop** command may be placed anywhere within the body of the **when** command.

**Syntax**

```
stop
```

**Arguments**

None.

Use the **run** command (CR-210) with the **-continue** option to continue the simulation run, or the **resume** command (CR-207) to continue macro execution. If you want macro execution to resume automatically, put the **resume** command at the top of your macro file:

```
onbreak {resume}
```

> **Note:** If you want to stop the simulation using a **when** command (CR-314), you must use a **stop** command within your when statement. **DO NOT** use an **exit** command (CR-152) or a **quit** command (CR-199). The **stop** command acts like a breakpoint at the time it is evaluated.

**See also**

bp (CR-68), resume (CR-207), run (CR-210), when (CR-314)
The `tb` (traceback) command displays a stack trace for the current process in the Main window. This lists the sequence of HDL function calls that have been entered to arrive at the current state for the active process.

**Syntax**

```
tb
```

**Arguments**

None.
toggle add

The `toggle add` command enables collection of toggle statistics for the specified nodes. The allowed nodes are Verilog nets and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored).

Syntax

toggle add
    

Arguments

- `-r`
  Specifies that toggle statistic collection is enabled recursively into subregions. Optional; if omitted, toggle statistic collection is limited to the current region.

- `-in`
  Enables toggle statistic collection on nodes of mode IN. Optional.

- `-out`
  Enables toggle statistic collection on nodes of mode OUT. Optional.

- `-inout`
  Enables toggle statistic collection on nodes of mode INOUT. Optional.

- `-internal`
  Enables toggle statistic collection on internal items. Optional.

- `-ports`
  Enables toggle statistic collection on nodes of modes IN, OUT, or INOUT. Optional.

- `<node_name>`
  Enables toggle statistic collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

toggle report (CR-226), toggle reset (CR-227)
**toggle report**

The **toggle report** command displays a list of all nodes that have not transitioned to both 0 and 1 at least once. Also displayed is a summary of the number of nodes checked, the number that toggled, the number that didn’t toggle, and a percentage that toggled.

**Syntax**

```
toggle report
   [-file <filename>] [-summary] [-all]
```

**Arguments**

- `file <filename>`
  Specifies a file to which to write the report. By default the report is displayed in the Main window. Optional.

- `summary`
  Selects only the summary portion of the report. Optional.

- `all`
  Lists all nodes checked along with their individual transition to 0 and 1 counts. Optional.

**See also**

`toggle add` (CR-225), `toggle reset` (CR-227)
toggle reset

The `toggle reset` command resets the toggle counts to zero for the specified nodes.

Syntax

```bash
toggle reset
   [-ports] <node_name>
```

Arguments

- `-all`
  Resets toggle statistic collection for all nodes that have toggle checking enabled. Optional. Must be used alone without other arguments.

- `-r`
  Specifies that toggle statistic collection is reset recursively into subregions. Optional; if omitted, the reset is limited to the current region.

- `-in`
  Resets toggle statistic collection on nodes of mode IN. Optional.

- `-out`
  Resets toggle statistic collection on nodes of mode OUT. Optional.

- `-inout`
  Resets toggle statistic collection on nodes of mode INOUT. Optional.

- `-internal`
  Resets toggle statistic collection on internal items. Optional.

- `-ports`
  Resets toggle statistic collection on nodes of modes IN, OUT, or INOUT. Optional.

- `<node_name>`
  Resets toggle statistic collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

`toggle add` (CR-225), `toggle report` (CR-226)
**transcribe**

The `transcribe` command displays a command in the Main window, then executes the command. The `transcribe` command is normally used to direct commands to the Main window from an external event such as a menu pick or button selection. The `add button` (CR-45) and `add_menuitem` (CR-54) commands can utilize `transcribe`. Returns nothing.

**Syntax**

```
transcribe
<command>
```

**Arguments**

`<command>`

Specifies the command to execute. Required.

**Examples**

```
add button pwd {transcribe pwd} NoDisable
```

Creates a button labeled "pwd" that invokes `transcribe` with the `pwd` Tcl command, and echoes the command and its results to the Main window. The button remains active during a run.

**See also**

`add button` (CR-45), `add_menuitem` (CR-54)
transcript

The transcript command controls echoing of commands executed in a macro file; it also works at top level in batch mode. If no option is specified, the current setting is reported.

Syntax

transcript
[on  | off  | -q  | quietly]

Arguments

on
  Specifies that commands in a macro file will be echoed to the Main window as they are executed. Optional.

off
  Specifies that commands in a macro file will not be echoed to the Main window as they are executed. Optional. The transcribe command (CR-228) can be used to force a command to be echoed.

-q
  Returns "0" if transcripting is turned off or "1" if transcripting is turned on. Useful in a Tcl conditional expression. Optional.

quietly
  Turns off the transcript echo for all commands. To turn off echoing for individual commands see the quietly command (CR-198). Optional.

Examples

transcript on
  Commands within a macro file will be echoed to the Main window as they are executed.

transcript
  If issued immediately after the previous example, the message:

  Macro transcripting is turned on.

  appears in the Main window.

See also

echo (CR-143), transcribe (CR-228)
tssi2mti

The **tssi2mti** command is used to convert a vector file in Fluence Technology (formerly TSSI) Standard Events Format into a sequence of **force** (CR-156) and **run** (CR-210) commands. The stimulus is written to the standard output.

The source code for **tssi2mti** is provided in the file `tssi2mti.c` in the `examples` directory.

Syntax

```bash
<signal_definition_file> [ <sef_vector_file> ]
```

Arguments

**<signal_definition_file>**
- Specifies the name of the Fluence Technology signal definition file describing the format and content of the vectors. Required.

**<sef_vector_file>**
- Specifies the name of the file containing vectors to be converted. If none is specified, standard input is used. Optional.

Examples

```bash
tssi2mti trigger.def trigger.sef > trigger.do
```
- The command will produce a do file named `trigger.do` from the signal definition file `trigger.def` and the vector file `trigger.sef`.

```bash
tssi2mti trigger.def < trigger.sef > trigger.do
```
- This example is exactly the same as the previous one, but uses the standard input instead.

See also

- **force** (CR-156), **run** (CR-210), **write tssi** (CR-329)
up

The **up** command searches for signal transitions or values in the specified List window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a signal takes on a particular value, or an expression of multiple signals evaluates to true. See the **down** command (CR-139) for related functionality.

The procedure for using **up** includes three steps: click on the desired signal; click on the desired starting location; issue the **up** command. (The **settime** command (CR-216) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

**Syntax**

```
up
  [-expr {<expression>}] [-falling] [-noglitch] [-rising]
  [-value <sig_value>] [-window <wname>] [<n>]
```

**Arguments**

- **-expr {<expression>}**
  The List window will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced List window. A signal may be specified either by its full path or by the shortcut label displayed in the List window.
  See "GUI_expression_format" (CR-18) for the format of the expression. The expression must be placed within curly braces.

- **-falling**
  Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

- **-noglitch**
  Specifies that delta-width glitches are to be ignored. Optional.

- **-rising**
  Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

- **-value <sig_value>**
  Specifies a value of the signal to match. Optional. Must be specified in the same radix that the selected signal is displayed. Case is ignored, but otherwise must be an exact string match -- don't-care bits are not yet implemented.

- **-window <wname>**
  Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-263) to change the default window.
<n>
  Specifies to find the nth match. Optional. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.

Examples

up -noglitch -value FF23
  Finds the last time at which the selected vector transitions to FF23, ignoring glitches.

up
  Goes to the previous transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-18) and can be built with the aid of the "The GUI Expression Builder" (UM-305).

up -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}
  Searches up for an expression that evaluates to a boolean 1 when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /top/u3/addr is equal to the specified 32-bit hex constant.

up -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
  Searches up for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal /top/u3/adder equals hex ac.

up -expr {{NOW > 23 us} && {NOW < 54 us}) && clk'rising && (mode == writing)}
  Searches up for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal mode is enumeration writing.

See also

"GUI_expression_format" (CR-18), view (CR-263), seetime (CR-216), down (CR-139)
vcd add

The **vcd add** command adds the specified items to a VCD file. The allowed items are Verilog nets and variables and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored).

All **vcd add** commands must be executed at the same simulation time. The specified items are added to the VCD header and their subsequent value changes are recorded in the specified VCD file.

By default all port driver changes and internal variable changes are captured in the file. You can filter the output using arguments detailed below.

Related Verilog tasks: $dumpvars, $fdumpvars

**Syntax**

```
vcd add
  <item_name>
```

**Arguments**

- `-r`
  Specifies that signal and port selection occurs recursively into subregions. Optional. If omitted, included signals and ports are limited to the current region.

- `-in`
  Includes only port driver changes from ports of mode IN. Optional.

- `-out`
  Includes only port driver changes from ports of mode OUT. Optional.

- `-inout`
  Includes only port driver changes from ports of mode INOUT. Optional.

- `-internal`
  Includes only internal variable or signal changes. Excludes port driver changes. Optional.

- `-ports`
  Includes only port driver changes. Excludes internal variable or signal changes. Optional.

- `-file <filename>`
  Specifies the name of the VCD file. This option should be used only when you have created multiple VCD files using the **vcd files** command (CR-245).

- `<item_name>`
  Specifies the Verilog or VHDL item to add to the VCD file. Required. Multiple items may be specified by separating names with spaces. Wildcards are accepted.

**See also**

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.
vcd checkpoint

The `vcd checkpoint` command dumps the current values of all VCD variables to the specified VCD file. While simulating, only value changes are dumped.

Related Verilog tasks: $dumpall, $fdumpall

Syntax

```
vcd checkpoint
[<filename>]
```

Arguments

`<filename>`
Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the `vcd file` command (CR-243) or "dump.vcd" if `vcd file` was not invoked.

See also

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files.
vcd comment

The `vcd comment` command inserts the specified comment in the specified VCD file.

Syntax

```
vcd comment
  <comment string>  [<filename>]
```

Arguments

- `<comment string>`
  Comment to be included in the VCD file. Required. Must be quoted by double quotation marks or curly braces.

- `<filename>`
  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the `vcd file` command (CR-243) or "dump.vcd" if `vcd file` was not invoked.

See also

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files.
vcd dumpports

The vcd dumpports command creates a VCD file that includes port driver data. By default all port driver changes and internal variable changes are captured in the file. You can filter the output using arguments detailed below.

Related Verilog task: $dumpports

Syntax

vcd dumpports

[-internal] [-ports] [-file <filename>] <item_name>

Arguments

- **-direction**
  Affects both VHDL and Verilog ports. Optional. Specifies that the port/variable type recorded in the VCD header for VHDL and Verilog ports shall be one of the following: in, out, inout, internal, ports (includes in, out, and inout); the default is all ports

  ▶️ **Note:** The -direction argument is obsolete in ModelSim versions 5.5c and later. It is supported for backwards compatibility only. See [http://www.model.com/products/documentation/resim_vcd.pdf](http://www.model.com/products/documentation/resim_vcd.pdf) for information regarding its use in earlier versions.

- **-map <mapping pairs>**
  Affects only VHDL signals of type std_logic. Optional. Overrides the default mappings. The mapping is specified as a list of character pairs. The first character in a pair must be one of the std_logic characters (U,X,0,1,Z,W,L,H,-), and the second character is the character you wish to be recorded in the VCD file. For example, to map L and H to z:

  ```
  vcd dumpports -map "L z H z"
  ```

  Note that the quotes in the example above are a Tcl convention for command strings that include spaces.

- **-nomap**
  Affects only VHDL signals of type std_logic. Optional. Specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of (U,X,0,1,Z,W,L,H,-). This option results in a non-standard VCD file because VCD values are limited to the four-state character set "x01z". By default, the std_logic characters are mapped as follows.

<table>
<thead>
<tr>
<th>VHDL</th>
<th>VCD</th>
<th>VHDL</th>
<th>VCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>x</td>
<td>W</td>
<td>x</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>x</td>
</tr>
</tbody>
</table>
vcd dumpports

Includes ports of mode IN. Optional.

-inout
Includes ports of mode INOUT. Optional.

-internal
Includes internal items. Excludes port driver changes. Optional.

-ports
Includes all ports of modes IN, OUT, or INOUT. Excludes internal variable or signal changes. Optional.

-file <filename>
Specifies the path and name of a VCD file to create. Optional. Defaults to the current working directory and the filename `dumpports.vcd`. Multiple filenames can be opened during a single simulation.

-<item_name>
Specifies the Verilog or VHDL item to add to the VCD file. Required. Multiple items may be specified by separating names with spaces. Wildcards are accepted.

Examples

```plaintext
vcd dumpports -in -file counter.vcd /test_counter/dut/*
Creates a VCD file named `counter.vcd` of all IN ports in the region `/test_counter/dut/`.

vcd dumpports -file addern.vcd /testbench/uut/*
vcddump -vcdstim addern.vcd addern -gn=8 -do "add wave */; run 1000"
These two commands resimulate a design from a VCD file. See "Resimulating a design from a VCD file" (UM-395) for further details.
```

See also

See Chapter 14 - Value Change Dump (VCD) Files for more information on VCD files.
vcd dumpportsall

The vcd dumpportsall command creates a checkpoint in the VCD file which shows the value of all selected ports at that time in the simulation, regardless of whether the port values have changed since the last timestep.

Related Verilog task: $dumpportsall

Syntax

vcd dumpportsall
   [<filename>]

Arguments

<filename>
   Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

See Chapter 14 - Value Change Dump (VCD) Files for more information on VCD files.
vcd dumpportsflush

The **vcd dumpportsflush** command flushes the contents of the VCD file buffer to the specified VCD file.

Related Verilog task: $dumpportsflush

**Syntax**

```vcd dumpportsflush [ <filename> ]```

**Arguments**

- `<filename>`
  Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

**See also**

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files.
vcd dumpportslimit

The `vcd dumpportslimit` command specifies the maximum size of the VCD file (by default, limited to available disk space). When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog task: `$dumpportslimit`

**Syntax**

```
vcd dumpportslimit
    <dumplimit> [<filename>]
```

**Arguments**

`<dumplimit>`

Specifies the maximum VCD file size in bytes. Required.

`<filename>`

Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

**See also**

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files.
vcd dumpportsoff

The `vcd dumpportsoff` command turns off VCD dumping and records all dumped port values as x.

Related Verilog task: `$dumpportsoff`

**Syntax**

```
vcd dumpportsoff
[<filename>]
```

**Arguments**

`<filename>`
Specify the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

**See also**

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files.
vcd dumpportson

The vcd dumpportson command turns on VCD dumping and records the current values of all selected ports. This command is typically used to resume dumping after invoking vcd dumpportsoff.

Related Verilog task: $dumpportson

Syntax

vcd dumpportson
    [<filename>]

Arguments

<filename>
    Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

See Chapter 14 - Value Change Dump (VCD) Files for more information on VCD files.
The `vcd file` command specifies the filename and state mapping for the VCD file created by a `vcd add` command (CR-233). The `vcd file` command is optional. If used, it must be issued before any `vcd add` commands.

Related Verilog task: `$dumpfile`

**Note:** `vcd file` is included for backward compatibility. Use the `vcd files` command (CR-245) if you want to use multiple VCD files during a single simulation.

### Syntax

```vcd file [-direction] [-dumpports] [<filename>] [-map <mapping pairs>] [-nomap]```

### Arguments

- **-direction**
  Affects only VHDL ports. Optional. It specifies that the port/variable type recorded in the VCD header for VHDL ports shall be one of the following:
  - in, out, inout, internal, ports (includes in, out, and inout); the default is all ports

  **Note:** The `-direction` argument is obsolete in ModelSim versions 5.5c and later. It is supported for backwards compatibility only. See [http://www.model.com/products/documentation/resim_vcd.pdf](http://www.model.com/products/documentation/resim_vcd.pdf) for information regarding its use in earlier versions.

- **-dumpports**
  Capture detailed port driver data for Verilog ports and VHDL std_logic ports. Optional. This option works only on ports, and subsequent `vcd add` command (CR-233) will accept only qualifying ports (silently ignoring all other specified items).

- **<filename>**
  Specifies the name of the VCD file that is created (the default is `dump.vcd`). Optional.

- **-map <mapping pairs>**
  Affects only VHDL signals of type std_logic. Optional. It allows you to override the default mappings. The mapping is specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. For example, to map L and H to Z:

  ```vcd file -map "L z H z"```

  Note that the quotes in the example above are a Tcl convention for command strings that include spaces.
Affects only VHDL signals of type std_logic. Optional. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

<table>
<thead>
<tr>
<th>VHDL</th>
<th>VCD</th>
<th>VHDL</th>
<th>VCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>x</td>
<td>W</td>
<td>x</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also

See Chapter 14 - Value Change Dump (VCD) Files for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.
**vcd files**

The **vcd files** command specifies a filename and state mapping for a VCD file created by a **vcd add** command (CR-233). The **vcd files** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $fdumpfile

**Syntax**

```
vcd files  
[-direction] <filename>  [-map <mapping pairs>] [-nomap]
```

**Arguments**

- **-direction**
  Affects both VHDL and Verilog ports. Optional. It specifies that the port/variable type recorded in the VCD header for VHDL and Verilog ports shall be one of the following: in, out, inout, internal, ports (includes in, out, and inout); the default is all ports

  ▶ **Note:** The -direction argument is obsolete in ModelSim versions 5.5c and later. It is supported for backwards compatibility only. See [http://www.model.com/products/documentation/resim_vcd.pdf](http://www.model.com/products/documentation/resim_vcd.pdf) for information regarding its use in earlier versions.

- **<filename>**
  Specifies the name of a VCD file to create. Required. Multiple files can be opened during a single simulation.

- **-map <mapping pairs>**
  Affects only VHDL signals of type std_logic. Optional. It allows you to override the default mappings. The mapping is specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. For example, to map L and H to z:

  ```
vcd files -map "L z H z"
  ```

  Note that the quotes in the example above are a Tcl convention for command strings that include spaces.
Affects only VHDL signals of type std_logic. Optional. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

<table>
<thead>
<tr>
<th>VHDL</th>
<th>VCD</th>
<th>VHDL</th>
<th>VCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>x</td>
<td>W</td>
<td>x</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.
### vcd flush

The **vcd flush** command flushes the contents of the VCD file buffer to the specified VCD file. This command is useful if you want to create a complete vcd file without ending your current simulation.

Related Verilog tasks: $dumpflush, $fdumpflush

#### Syntax

```
vcd flush
[<filename>]
```

#### Arguments

**<filename>**

Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the **vcd file** command (CR-243) or dump.vcd if **vcd file** was not invoked.

#### See also

See Chapter 14 - Value Change Dump (VCD) Files for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.
vcd limit

The vcd limit command specifies the maximum size of a VCD file (by default, limited to available disk space). When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog tasks: $dumplimit, $fdumplimit

Syntax

vcd limit
  <filesize>  [ <filename> ]

Arguments

<filesize>
  Specifies the maximum VCD file size in bytes. Required.

<filename>
  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the vcd file command (CR-243) or dump.vcd if vcd file was not invoked.

See also

See Chapter 14 - Value Change Dump (VCD) Files for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.
vcd off

The `vcd off` command turns off VCD dumping to the specified file and records all VCD variable values as x.

Related Verilog tasks: `$dumpoff`, `$fdumpoff`

**Syntax**

```
vcd off
[<filename>]
```

**Arguments**

- `<filename>`
  - Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the `vcd file` command (CR-243) or `dump.vcd` if `vcd file` was not invoked.

**See also**

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.
**vcd on**

The `vcd on` command turns on VCD dumping to the specified file and records the current values of all VCD variables. By default, `vcd on` is automatically performed at the end of the simulation time that the `vcd add` (CR-233) commands are performed.

Related Verilog tasks: `$dump`, `$fdump`

**Syntax**

```
vcd on
[<filename>]
```

**Arguments**

`<filename>`

Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the `vcd file` command (CR-243) or `dump.vcd` if `vcd file` was not invoked.

**See also**

See *Chapter 14 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog system tasks are documented in the IEEE 1364 standard.
vcd2wlf

vcd2wlf is a utility that translates a VCD (Value Change Dump) file into a WLF file that can be displayed in ModelSim using the `vsim -view` argument.

Syntax

```
vcd2wlf  <vcd filename> <wlf filename>
```

Arguments

- `<vcd filename>`
  Specifies the name of the VCD file you want to translate into a WLF file. Required.

- `<wlf filename>`
  Specifies the name of the output WLF file. Required.
vcom

The vcom command is used to invoke VCOM, the Model Technology VHDL compiler. Use VCOM to compile VHDL source code into a specified working library (or to the work library by default).

This command may be invoked from within ModelSim or from the operating system command prompt. This command may also be invoked during simulation.

Syntax

```
```

Arguments

-87
Disables support for VHDL 1076-1993. This is the VCOM default. Optional. See additional discussion in the examples. Note that the default can be changed with the `modelsim.ini` file; see "Preference variables located in INI files" (UM-444).

-93
Specifies that the simulator is to support VHDL 1076-1993. Optional. Default is -87. See additional discussion in the examples.

-check_synthesis
Turns on limited synthesis rule compliance checking; specifically, it checks to see that signals read by a process are in the sensitivity list. Optional. The checks understand only combinatorial logic, not clocked logic. Edit the `CheckSynthesis` (UM-445) variable in the `modelsim.ini` file to set a permanent default.

-debugVA
Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration. Optional.

-defercheck
Defers until run-time all compile-time range checking on constant index and slice expressions. As a result, index and slice expressions with invalid constant ranges that are never evaluated will not cause compiler error messages to be issued. Optional.

-explicit
Directs the compiler to resolve ambiguous function overloading by favoring the explicit function definition over the implicit function definition. Optional. Strictly speaking, this behavior does not match the VHDL standard. However, the majority of EDA tools
choose explicit operators over implicit operators. Using this switch makes ModelSim compatible with common industry practice.

\[-f \text{ <filename>}\]
Specifies a file with more command line arguments. Allows complex arguments to be reused without retying. Optional.

\[-force\_refresh\]
Forces the refresh of a module. Optional. When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. If a dependency has been changed and recompiled, the compiler will not refresh the dependent design unit (unless you use \[-force\_refresh\]). To avoid potential errors or mismatches caused by the dependency recompilation, you should recompile the dependent design unit’s source rather than use this switch.

\[-help\]
Displays the command’s options and arguments. Optional.

\[-ignoredefaultbinding\]
Instructs the compiler not to generate a default binding during compilation. Optional. You must explicitly bind all components in the design to use this switch.

\[-ignorevitalerrors\]
Directs the compiler to ignore VITAL compliance errors. Optional. The compiler still reports that VITAL errors exist, but it will not stop the compilation. You should exercise caution in using this switch; as part of accelerating VITAL packages, we assume that compliance checking has passed.

\[-just\ abcep\]
Directs the compiler to “just” include:
\(a\) - architectures
\(b\) - bodies
\(c\) - configurations
\(e\) - entities
\(p\) - packages

Any combination in any order can be used, but one choice is required if you use this optional switch.

\[-skip\ abcep\]
Directs the compiler to skip all:
\(a\) - architectures
\(b\) - bodies
\(c\) - configurations
\(e\) - entities
\(p\) - packages

Any combination in any order can be used, but one choice is required if you use this optional switch.

\[-line \text{ <number>}\]
Starts the compiler on the specified line in the VHDL source file. Optional. By default, the compiler starts at the beginning of the file.
-no1164
Causes the source files to be compiled without taking advantage of the built-in version of the IEEE std_logic_1164 package. Optional. This will typically result in longer simulation times for VHDL programs that use variables and signals of type std_logic.

-noaccel <package_name>
Turns off acceleration of the specified package in the source code using that package.

-nocasestaticerror
Suppresses case static warnings. Optional. VHDL standards require that case alternative choices be static at compile time. However, some expressions which are globally static are allowed. This switch prevents the compiler from warning on such expressions. If the -pedanticerrors switch is specified, this switch is ignored.

-nocheck
Disables index and range checks. Optional. You can disable these individually using the -noindexcheck and -norangecheck arguments, respectively.

-nodebug[=ports]
Hides the internal data of the compiled design unit. Optional. The design unit’s source code, internal structure, signals, processes, and variables will not display in ModelSim’s windows. In addition, none of the hidden objects may be accessed through the Dataflow window or with commands. This also means that you cannot set breakpoints or single step within this code. Don’t compile with this switch until you’re done debugging.

Note that this is not a speed switch like the “nodebug” option on many other products.

The optional =ports switch hides the ports for the lower levels of your design; it should only be used to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

► Note: -nodebug provides protection for proprietary model information. The Verilog `protect compiler directive provides similar protection, but uses a Cadence encryption algorithm that is unavailable to Model Technology.

Design units or modules compiled with -nodebug can only instantiate design units or modules that are also compiled -nodebug.

See additional discussion in "Source code security and -nodebug" (UM-492).

-noindexcheck
Disables checking on indexing expressions to determine whether indices are within declared array bounds. Optional.

-nologo
Disables startup banner. Optional.

-noothersstaticerror
Disables warnings that result from array aggregates with multiple choices having "others" clauses that are not locally static. Optional. If the -pedanticerrors switch is specified, this switch is ignored.

-norangecheck
Disables run time range checking. In some designs, this results in a 2X speed increase. Range checking is enabled by default or, once disabled, can be enabled using -rangecheck.
-novital
Causes VCOM to use VHDL code for VITAL procedures rather than the accelerated and optimized timing and primitive packages built into the simulator kernel. Optional.

-nowitalcheck
Disables VITAL 2000 compliance checking if you are using VITAL 2.2b. Optional.

-nowarn <number>
Selectively disables an individual warning message. Optional. Multiple -nowarn switches are allowed. Warnings may be disabled for all compiles via the modelsim.ini file (see the "[vcom] VHDL compiler control variables" (UM-445)).

The warning message numbers are:
1 = unbound component
2 = process without a wait statement
3 = null range
4 = no space in time literal
5 = multiple drivers on unresolved signal
6 = compliance checks
7 = optimization messages

-O0 | -O1 | -O4 | -O5
Lower the optimization to a minimum with -O0 (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Enable PE-level optimization with -O1. Optional. Note that changing from the default -O4 to -O1 may cause event order differences in your simulation.

Enable standard SE optimizations with -O4. Default. The main differences between -O4 and -O1 is that ModelSim attempts to improve memory management for vectors and accelerate VITAL Level 1 modules with -O4.

Enable maximum optimization with -O5. Optional. The main difference between -O5 and -O4 is ModelSim attempts to optimize loops with -O5. We recommend use of this switch with large sequential blocks only; other uses may significantly increase compile times.

-pedanticerrors
Forces ModelSim to error (rather than warn) on two conditions: 1) when a choice in a case statement is not a locally static expression; 2) when an array aggregate with multiple choices doesn’t have a locally static "others" choice. Optional. This argument overrides -nocasestaticerror and -noothersstaticerror (see above).

-performdefaultbinding
Enables default binding when it has been disabled via the
RequireConfigForAllDefaultBinding option in the modelsim.ini file. Optional.

-quiet
Disable 'loading' messages. Optional.

-rangecheck
Enables run time range checking. Default. Range checking can be disabled using the -norangecheck argument.
-refresh
   Regenerates a library image. Optional. By default, the work library is updated; use -work
   <library> to update a different library. See vcom "Examples" (CR-257) for more
   information.

-s
   Instructs the compiler not to load the standard package. Optional. This argument should
   only be used if you are compiling the standard package itself.

-source
   Displays the associated line of source code before each error message that is generated
   during compilation. Optional; by default, only the error message is displayed.

-time
   Reports the "wall clock time" vcom takes to compile the design. Optional. Note that if
   many processes are running on the same system, wall clock time may differ greatly from
   the actual "cpu time" spent on vcom.

-version
   Returns the version of the compiler as used by the licensing tools, such as "Model
   Technology ModelSim SE vcom 5.5 Compiler 2000.01 Jan 29 2000".

-work <library_name>
   Specifies a logical name or pathname of a library that is to be mapped to the logical
   library work. Optional; by default, the compiled design units are added to the work
   library. The specified pathname overrides the pathname specified for work in the project
   file.

-filename
   Specifies the name of a file containing the VHDL source to be compiled. One filename
   is required; multiple filenames can be entered separated by spaces or wildcards may be
   used (e.g., *.vhd).
Examples

vcom example.vhd

Compiles the VHDL source code contained in the file example.vhd.

vcom -87 o_units1.vhd o_units2.vhd
vcom -93 n_unit91.vhd n_unit92.vhd

ModelSim supports designs that use elements conforming to both the 1993 and the 1987 standards. Compile the design units separately using the appropriate switches.

Note that in the example above, the -87 switch on the first line is redundant since the VCOM default is to compile to the 1987 standard.

vcom -nodebug example.vhd

Hides the internal data of example.vhd. Models compiled with -nodebug cannot use any of the ModelSim debugging features; any subsequent user will not be able to see into the model.

vcom -nodebug=ports level3.vhd leve2.vhd
vcom -nodebug top.vhd

The first line compiles and hides the internal data, plus the ports, of the lower-level design units, level3.vhd and level2.vhd. The second line compiles the top-level unit, top.vhd, without hiding the ports. It is important to compile the top level without =ports because top-level ports must be visible for simulation.

See "Source code security and -nodebug" (UM-492) for more details.

vcom -noaccel numeric_std example.vhd

When compiling source that uses the numeric_std package, this command turns off acceleration of the numeric_std package, located in the ieee library.

vcom -explicit example.vhd

Although it is not obvious, the = operator is overloaded in the std_logic_1164 package. All enumeration data types in VHDL get an “implicit” definition for the = operator. So while there is no explicit = operator, there is an implicit one. This implicit declaration can be hidden by an explicit declaration of = in the same package (LRM Section 10.3). However, if another version of the = operator is declared in a different package than that containing the enumeration declaration, and both operators become visible through use clauses, neither can be used without explicit naming.

ARITHMETIC.=left, right

To eliminate that inconvenience, the VCOM command has the -explicit option that allows the explicit = operator to hide the implicit one. Allowing the explicit declaration to hide the implicit declaration is what most VHDL users expect.

vcom -work mylib -refresh

The -work option specifies mylib as the library to regenerate. -refresh rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim (4.6 and later only).

If your library contains Verilog design units be sure to regenerate the library with vlog (CR-288) and -refresh as well. See "Regenerating your design libraries" (UM-55) for more information.
**vdel**

The `vdel` command deletes a design unit from a specified library.

**Syntax**

```
vdel [-help] [-verbose] [-lib <library_name>] [-all | <design_unit> [<arch_name>]]
```

**Arguments**

- `-help`
  Displays the command’s options and arguments. Optional.

- `-verbose`
  Displays progress messages. Optional.

- `-lib <library_name>`
  Specifies the logical name or pathname of the library that holds the design unit to be deleted. Optional; by default, the design unit is deleted from the `work` library.

- `-all`
  Deletes an entire library. Optional. BE CAREFUL! Libraries cannot be recovered once deleted, and you are not prompted for confirmation.

- `<design_unit>`
  Specifies the entity, package, configuration, or module to be deleted. Required unless `-all` is used.

- `<arch_name>`
  Specifies the name of an architecture to be deleted. Optional; if omitted, all of the architectures for the specified entity are deleted. Invalid for a configuration or a package.

**Examples**

```
vdel -all
  Deletes the `work` library.

vdel -lib synopsys -all
  Deletes the `synopsys` library.

vdel xor
  Deletes the entity named `xor` and all its architectures from the `work` library.

vdel xor behavior
  Deletes the architecture named `behavior` of the entity `xor` from the `work` library.

vdel base
  Deletes the package named `base` from the `work` library.
```
vdir

The `vdir` command selectively lists the contents of a design library. This command can also be used to check compatibility of a vendor library. If vdir cannot read a vendor-supplied library, the library may not be ModelSim compatible.

Syntax

```plaintext
vdir
   [-help] [-l] [-r] [-lib <library_name>] [<design_unit>]
```

Arguments

- `-help`
  Displays the command’s options and arguments. Optional.

- `-l`
  Prints the version of `vcom` or `vlog` that each design unit was compiled under. Also prints the object-code version number that indicates which versions of `vcom/vlog` and ModelSim are compatible. This example was printed by `vdir -l` for the counter module in the `work` library:

  ```
  # MODULE counter
  # Verilog Version: OzO;ZAV1R1jO;>KYtq2kY2
  # Source directory: ..\examples\projects\mixed
  # Source modified time: 944001078
  # Source file: ../examples/projects/verilog/counter.v
  # Opcode format: 5.4 Beta 4; VLOG EE Object version 17
  # Version number: e:VQh7zF_VJYN9MbEXUG_3
  # Optimized Verilog design root: 1
  # Language standard: 1
  ```

- `-r`
  Prints architecture information for each entity in the output.

- `-lib <library_name>`
  Specifies the logical name or the pathname of the library to be listed. Optional; by default, the contents of the `work` library are listed.

- `<design_unit>`
  Indicates the design unit to search for within the specified library. If the design unit is a VHDL entity, its architectures are listed. Optional; by default, all entities, configurations, modules, and packages in the specified library are listed.

Example

```plaintext
vdir -lib design my_asic
Lists the architectures associated with the entity named `my_asic` that reside in the HDL design library called `design`.
```
verror

The verror command prints a detailed description about a message number. It may also point to additional documentation related to the error.

Syntax

```plaintext
verror
<msgNum>...
```

Arguments

- `<msgNum>`
  Specifies the message number of a ModelSim message. Required. This number can be obtained from messages that have the format:

```plaintext
** <Level>: ([<Tool>-[<Group>-]<MsgNum>) <FormattedMsg>
```

Example

Say you see the following message in the transcript:

```plaintext
** Error (vsim-3601) foo.v(22): Too many Verilog port connections.
```

You would type:

```plaintext
verror 3061
```

and receive the following output:

```plaintext
Message # 3061:
Too many Verilog ports were specified in a mixed VHDL/Verilog instantiation. Verify that the correct VHDL/Verilog connection is being made and that the number of ports matches.

[DOC: ModelSim User’s Manual - Mixed VHDL and Verilog Designs Chapter]
```
Once a Verilog module is compiled into a library, you can use the `vgencomp` command to write its equivalent VHDL component declaration to standard output. Optional switches allow you to generate bit or vl_logic port types; std_logic port types are generated by default.

**Syntax**

```plaintext
vgencomp
   [-help] [-lib <library_name>] [-b] [-s] [-v] <module_name>
```

**Arguments**

- `-help`
  Displays the command’s options and arguments. Optional.

- `-lib <library_name>`
  Specifies the pathname of the working library. If not specified, the default library `work` is used. Optional.

- `-b`
  Causes `vgencomp` to generate bit port types. Optional.

- `-s`
  Used for the explicit declaration of default std_logic port types. Optional.

- `-v`
  Causes `vgencomp` to generate vl_logic port types. Optional.

- `<module_name>`
  Specifies the name of the Verilog module to be accessed. Required.

**Examples**

This example uses a Verilog module that is compiled into the `work` library. The module begins as Verilog source code:

```verilog
module top(i1, o1, o2, io1);
   parameter width = 8;
   parameter delay = 4.5;
   parameter filename = "file.in";
   input i1;
   output [7:0] o1;
   output [4:7] o2;
   inout [width-1:0] io1;
endmodule
```

After compiling, `vgencomp` is invoked on the compiled module:

```plaintext
vgencomp top
```

and writes the following to stdout:

```plaintext
component top
generic(
```
width           : integer  := 8;
delay           : real      := 4.500000;
filename        : string    := "file.in"
);
port(
    i1              : in    std_logic;
    o1              : out   std_logic_vector(7 downto 0);
    o2              : out   std_logic_vector(4 to 7);
    io1             : inout std_logic_vector
    iol             : inout std_logic_vector
);
end component;
view

The `view` command will open a ModelSim window and bring that window to the front of the display. If multiple instances of a window exist, `view` will change the default window of that type to the specified window. Using the `-new` option, `view` will create an additional instance of the specified window type and set it to be the default window for that type.

Names for windows are generated as follows:

- The first window name (automatically created without using `-new`) has the same name as the window type.
- Additional window names created by `-new` append an integer to the window type, starting with 1.

To remove a window, use the `noview` command (CR-177).

Syntax

```
view [*] [-height <n>] [-icon] [-new] [-title {New Window Title} <window_type>] [-width <n>] [-x <n>] [-y <n>] <window_type>...
```

Arguments

* Specifies that all windows be opened. Optional.
- `-height <n>` Specifies the window height in pixels. Optional.
- `-icon` Toggles the view between window and icon. Optional.
- `-new` Creates a new instance of the window type specified with the `<window_type>` argument. Optional. New window names are created by appending an integer to the window type, starting with 1, then incrementing the integer.
- `-title {New Window Title} <window_type>` Specifies the window title of the designated window. Curly braces are only needed for titles that include spaces. Double quotes can be used in place of braces, for example "New Window Title". If the new window title does not include spaces, no braces or quotes are needed. For example: `-title new_wave wave` assigns the title `new_wave` to the Wave window.
- `-width <n>` Specifies the window width in pixels. Optional.

Available window types are:

- dataflow, list, process, signals, source, structure, variables, and wave
Also creates a new instance of the specified window type when used with the -new option. You may also specify the window(s) to view when multiple instances of that window type exist (e.g., wave2, structure1).

-x <n>
Specifies the window upper-left-hand x-coordinate in pixels. Optional.

-y <n>
Specifies the window upper-left-hand y-coordinate in pixels. Optional.

Examples

view d
  Opens the Dataflow window.

view si pr
  Opens the Signals and Process windows.

view s
  Opens the Signals, Source, and Structure windows.

view -title {My Wave Window} wave
  Opens a new wave window with My Wave Window as its title.

view wave
view -new wave
  The first command creates a window named 'wave'. The second command creates a window named 'wave1'. Its full Tk path is '.wave1'. Wave1 is now the default Wave window. Any add wave command (CR-57) would add items to wave1.

view wave
  Changes the default Wave window back to 'wave'.

add wave -win .wave1 mysig
  Will override the default Wave window and add mysig to wave1.

See also

noview (CR-177)
virtual count

The **virtual count** command counts the number of currently defined virtuals that were not read in using a macro file.

**Syntax**

```
virtual count
    [-kind <kind>]
```

**Arguments**

- **-kind <kind>**
  Specifies a subset of virtuals to look at. Optional. `<kind>` can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- **-unsaved**
  Specifies that the count include only those virtuals that have not been saved. Optional.

**See also**

- virtual define (CR-266), virtual save (CR-280), virtual show (CR-281), "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual define

The `virtual define` command prints to the Main window the definition of the virtual signal or function in the form of a command that can be used to re-create the object.

Syntax

```
virtual define
    [-kind <kind>] <pathname>
```

Arguments

- `-kind <kind>`
  Specifies a subset of virtuals to look at. Optional. `<kind>` can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- `<pathname>`
  Specifies the path to the virtual(s) for which you want definitions. Required. Wildcards can be used.

Examples

```
virtual define -kind explicits *
```

Shows the definitions of all the virtuals you have explicitly created.

See also

- `virtual describe` (CR-268)
- `virtual show` (CR-281)
- "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual delete

The virtual delete command removes the matching virtuals.

Syntax

virtual delete
   [-kind <kind>] <pathname>

Arguments

- -kind <kind>
   Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- <pathname>
   Specifies the path to the virtual(s) you want to delete. Required. Wildcards can be used.

Examples

virtual delete -kind explicits *
   Deletes all of the virtuals you have explicitly created.

See also

virtual signal (CR-282), virtual function (CR-270), "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual describe

The virtual describe command prints to the Main window a complete description of the data type of one or more virtual signals. Similar to the existing describe command.

Syntax

```
virtual describe
    [-kind <kind>] <pathname>
```

Arguments

- **-kind <kind>**
  Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- **<pathname>**
  Specifies the path to the virtual(s) for which you want descriptions. Required. Wildcards can be used.

Examples

```
virtual describe -kind explicits *

Describes the data type of all virtuals you have explicitly created.
```

See also

virtual define (CR-266), virtual show (CR-281), "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual expand

The virtual expand command produces a list of all the non-virtual objects contained in the specified virtual signal(s). This can be used to create a list of arguments for a command that does not accept or understand virtual signals.

Syntax

virtual expand
   [-base] <pathname>

Arguments

- **-base**
  Causes the root signal parent to be output in place of a subelement. Optional. For example:
  
  vcd add [virtual expand -base myVirtualSignal]

  the resulting command after substitution would be:
  
  vcd add signala signalb signalc

- **<pathname>**
  Specifies the path to the signals and virtual signals to expand. Required. Wildcards can be used. Any number of paths can be specified.

Examples

vcd add [virtual expand myVirtualSignal]

Adds the elements of a virtual signal to the VCD file.

In the Tcl language, the square brackets specify that the enclosed command should be executed first ("virtual expand ..."), then the result substituted into the surrounding command. So if myVirtualSignal is a concatenation of signala, signalb.rec1 and signalc(5 downto 3), the resulting command after substitution would be:

vcd add signala signalb.rec1 {signalc(5 downto 3)}

The slice of signalc is quoted in curly braces, because it contains spaces.

See also

virtual signal (CR-282), "Virtual Objects (User-defined buses, and more)" (UM-161)
The **virtual function** command creates a new signal, known only by the GUI (not the kernel), that consists of logical operations on existing signals and simulation time, as described in `<expressionString>`. It cannot handle bit selects and slices of Verilog registers. Please see "Syntax and conventions" (CR-9) for more details on syntax.

If the virtual function references more than a single scalar signal, it will display as an expandable object in the Wave and Signals windows. The children correspond to the inputs of the virtual function. This allows the function to be "expanded" in the Wave window to see the values of each of the input waveforms, which could be useful when using virtual functions to compare two signal values.

Virtual functions can also be used to gate the List window display.

**Syntax**

```plaintext
virtual function
   [-env <path>] [-install <path>] [-implicit] [-delay <time>]
   {<expressionString>} <name>
```

**Arguments**

Arguments for **virtual function** are the same as those for **virtual signal**, except for the contents of the expression string.

- **-env <path>**
  Specifies a hierarchical context for the signal names in `<expressionString>` so they don’t all have to be full paths. Optional.

- **-install <path>**
  Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in `<expressionString>`. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region `virtuals:/Functions`. Optional.

- **-implicit**
  Used internally to create virtuals that are automatically saved with the List or Wave format. Optional.

- **-delay <time>**
  Specifies a value by which the virtual function will be delayed. Optional. You can use negative values to look forward in time. If units are specified, the `<time>` option must be enclosed in curly braces. See the examples below for more details.

- `{<expressionString>}`
  A text string expression in the MTI GUI expression format. Required. See "GUI_expression_format" (CR-18) for more information.
virtual function

<name>
The name you define for the virtual signal. Required. Case is ignored unless installed in a Verilog region. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, <name> needs to be quoted with double quotes or with curly braces.

Examples

virtual function { not /chip/section1/clk } clk_n
Creates a signal /chip/section1/clk_n that is the inverse of /chip/section1/clk.

virtual function -install /chip { (std_logic_vector) chip.vlog.rega } rega_slv
Creates a std_logic_vector equivalent of a verilog register rega and installs it as /chip/rega_slv.

virtual function { /chip/addr[11:0] == 0xfab } addr_eq_fab
Creates a boolean signal /chip/addr_eq_fab that is true when /chip/addr[11:0] is equal to hex "fab", and false otherwise. It is acceptable to mix VHDL signal path notation with Verilog part-select notation.

virtual function { gate:/chip/siga XOR rtl:/chip/siga) } siga_diff
Creates a signal that is high only during times when signal /chip/siga of the gate-level version of the design does not match /chip/siga of the rtl version of the design. Because there is no common design region for the inputs to the expression, siga_diff is installed in region virtuals:/Functions. The virtual function siga_diff can be added to the Wave window, and when expanded will show the two original signals that are being compared.

virtual function -delay {10 ns} {/top/signalA AND /top/signalB} myDelayAandB
Creates a virtual signal consisting of the logical "AND" function of /top/signalA with /top/signalB, and delays it by 10 ns.

virtual function { | (gate:/chip/outbus XOR rtl:/chip/outbus) } outbus_diff
Creates a one-bit signal outbus_diff which is non-zero during times when any bit of /chip/outbus in the gate-level version doesn’t match the corresponding bit in the rtl version.

This expression uses the "OR-reduction" operator, which takes the logical OR of all the bits of the vector argument.

Commands fully compatible with virtual functions

<table>
<thead>
<tr>
<th>add list (CR-48)</th>
<th>add log /log (CR-166)</th>
<th>add wave (CR-57)</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkpoint (CR-82) and restore (CR-206)</td>
<td>delete (CR-133)</td>
<td>describe (CR-134) (&quot;virtual describe&quot; is a little faster)</td>
</tr>
<tr>
<td>down (CR-139) / up (CR-231)</td>
<td>examine (CR-149)</td>
<td>find (CR-153)</td>
</tr>
<tr>
<td>restart (CR-204)</td>
<td>left (CR-164) / right (CR-208)</td>
<td>search (CR-212)</td>
</tr>
<tr>
<td>searchlog (CR-214)</td>
<td>show (CR-218)</td>
<td></td>
</tr>
</tbody>
</table>
### Commands not currently compatible with virtual functions

<table>
<thead>
<tr>
<th>Command</th>
<th>CR-Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>check contention add</td>
<td>CR-74</td>
</tr>
<tr>
<td>check contention config</td>
<td>CR-75</td>
</tr>
<tr>
<td>check contention off</td>
<td>CR-76</td>
</tr>
<tr>
<td>check float add</td>
<td>CR-77</td>
</tr>
<tr>
<td>check float config</td>
<td>CR-78</td>
</tr>
<tr>
<td>check float off</td>
<td>CR-79</td>
</tr>
<tr>
<td>check stable on</td>
<td>CR-81</td>
</tr>
<tr>
<td>check stable off</td>
<td>CR-80</td>
</tr>
<tr>
<td>drivers</td>
<td>CR-141</td>
</tr>
<tr>
<td>force</td>
<td>CR-156</td>
</tr>
<tr>
<td>noforce</td>
<td>CR-173</td>
</tr>
<tr>
<td>power add</td>
<td>CR-184</td>
</tr>
<tr>
<td>power report</td>
<td>CR-185</td>
</tr>
<tr>
<td>power reset</td>
<td>CR-186</td>
</tr>
<tr>
<td>toggle add</td>
<td>CR-225</td>
</tr>
<tr>
<td>toggle reset</td>
<td>CR-227</td>
</tr>
<tr>
<td>toggle report</td>
<td>CR-226</td>
</tr>
<tr>
<td>virtual count</td>
<td>CR-265</td>
</tr>
<tr>
<td>virtual define</td>
<td>CR-266</td>
</tr>
<tr>
<td>virtual delete</td>
<td>CR-267</td>
</tr>
<tr>
<td>virtual describe</td>
<td>CR-268</td>
</tr>
<tr>
<td>virtual expand</td>
<td>CR-269</td>
</tr>
<tr>
<td>virtual hide</td>
<td>CR-273</td>
</tr>
<tr>
<td>virtual log</td>
<td>CR-274</td>
</tr>
<tr>
<td>virtual nohide</td>
<td>CR-276</td>
</tr>
<tr>
<td>virtual nolog</td>
<td>CR-277</td>
</tr>
<tr>
<td>virtual region</td>
<td>CR-279</td>
</tr>
<tr>
<td>virtual save</td>
<td>CR-280</td>
</tr>
<tr>
<td>virtual show</td>
<td>CR-281</td>
</tr>
<tr>
<td>virtual signal</td>
<td>CR-282</td>
</tr>
<tr>
<td>virtual type</td>
<td>CR-285</td>
</tr>
</tbody>
</table>

### See also

<table>
<thead>
<tr>
<th>Command</th>
<th>CR-Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual count</td>
<td>CR-265</td>
</tr>
<tr>
<td>virtual define</td>
<td>CR-266</td>
</tr>
<tr>
<td>virtual delete</td>
<td>CR-267</td>
</tr>
<tr>
<td>virtual describe</td>
<td>CR-268</td>
</tr>
<tr>
<td>virtual expand</td>
<td>CR-269</td>
</tr>
<tr>
<td>virtual hide</td>
<td>CR-273</td>
</tr>
<tr>
<td>virtual log</td>
<td>CR-274</td>
</tr>
<tr>
<td>virtual nohide</td>
<td>CR-276</td>
</tr>
<tr>
<td>virtual nolog</td>
<td>CR-277</td>
</tr>
<tr>
<td>virtual region</td>
<td>CR-279</td>
</tr>
<tr>
<td>virtual save</td>
<td>CR-280</td>
</tr>
<tr>
<td>virtual show</td>
<td>CR-281</td>
</tr>
<tr>
<td>virtual signal</td>
<td>CR-282</td>
</tr>
<tr>
<td>virtual type</td>
<td>CR-285</td>
</tr>
<tr>
<td>Virtual Objects (User-defined buses, and more)</td>
<td>CR-161</td>
</tr>
</tbody>
</table>
virtual hide

The **virtual hide** command sets a flag in the specified real or virtual signals, so those signals do not appear in the Signals window. This is used when you want to replace an expanded bus with a user-defined bus. You make the signals reappear using the **virtual nohide** command.

Syntax

```
virtual hide
    [ -kind <kind> ]|[-region <path>] <pattern>
```

Arguments

- **-kind <kind>**
  Specifies a subset of virtuals to look at. Optional. `<kind>` can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- **-region <path>**
  Used in place of -kind to specify a region of design space in which to look for the signal names. Optional.

- **<pattern>**
  Indicates which signal names or wildcard patterns should be used in finding the signals to hide. Required. Any number of names or wildcard patterns may be used.

See also

- **virtual nohide** (CR-276), "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual log

The virtual log command causes the simulation-mode dependent signals of the specified virtual signals to be logged by the kernel. If wildcard patterns are used, it will also log any normal signals found, unless the -only option is used. You unlog the signals using the virtual nolog command.

Syntax

virtual log

Arguments

- kind <kind>
  Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- region <path>
  Used in place of -kind to specify a region of design space in which to look for signals to log. Optional.

- recursive
  Specifies that the scope of the search is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region.

- only
  Can be used with a wildcard to specify that only virtual signals (as opposed to all signals) found by the wildcard should be logged. Optional.

- in
  Specifies that the kernel log data for ports of mode IN whose names match the specification. Optional.

- out
  Specifies that the kernel log data for ports of mode OUT whose names match the specification. Optional.

- inout
  Specifies that the kernel log data for ports of mode INOUT whose names match the specification. Optional.

- internal
  Specifies that the kernel log data for internal items whose names match the specification. Optional.

- ports
  Specifies that the kernel log data for all ports. Optional.

<pattern>
Indicates which signal names or wildcard patterns should be used in finding the signals to log. Required. Any number of names or wildcard patterns may be used.
See also

virtual nolog (CR-277). "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual nohide

The **virtual nohide** command reverses the effect of a **virtual hide** command. It resets the flag in the specified real or virtual signals, so those signals reappear in the Signals window.

Syntax

```
virtual nohide
   [-kind <kind>] [-region <path>] <pattern>
```

Arguments

- **-kind <kind>**
  Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- **-region <path>**
  Used in place of -kind to specify a region of design space in which to look for the signal names. Optional.

- **<pattern>**
  Indicates which signal names or wildcard patterns should be used in finding the signals to expose. Required. Any number of names or wildcard patterns may be used.

See also

**virtual hide** (CR-273), "Virtual Objects (User-defined buses, and more)" (UM-161)
The virtual nolog command reverses the effect of a virtual log command. It causes the simulation-dependent signals of the specified virtual signals to be excluded ("unlogged") by the kernel. If wildcard patterns are used, it will also unlog any normal signals found, unless the -only option is used.

Syntax

```
virtual nolog
[-internal] [-ports] <pattern>
```

Arguments

- **-kind <kind>**
  Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- **-region <path>**
  Used in place of -kind to specify a region of design space in which to look for signals to unlog. Optional.

- **-recursive**
  Specifies that the scope of the search is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region.

- **-only**
  Can be used with a wildcard to specify that only virtual signals (as opposed to all signals) found by the wildcard should be unlogged. Optional.

- **-in**
  Specifies that the kernel exclude data for ports of mode IN whose names match the specification. Optional.

- **-out**
  Specifies that the kernel exclude data for ports of mode OUT whose names match the specification. Optional.

- **-inout**
  Specifies that the kernel exclude data for ports of mode INOUT whose names match the specification. Optional.

- **-internal**
  Specifies that the kernel exclude data for internal items whose names match the specification. Optional.

- **-ports**
  Specifies that the kernel exclude data for all ports. Optional.

- **<pattern>**
  Indicates which signal names or wildcard pattern should be used in finding the signals to unlog. Required. Any number of names or wildcard patterns may be used.
See also

virtual log (CR-274), "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual region

The virtual region command creates a new user-defined design hierarchy region.

Syntax

virtual region
   <parentPath>  <regionName>

Arguments

<parentPath>
The full path to the region that will become the parent of the new region. Required.

<regionName>
The name you want for the new region. Required.

See also

virtual function (CR-270), virtual signal (CR-282), "Virtual Objects (User-defined buses, and more)" (UM-161)

► Note: Virtual regions cannot be used in the when (CR-314) command.
virtual save

The `virtual save` command saves the definitions of virtuals to a file.

Syntax

```
virtual save
 [-kind <kind>] [-append] [<filename>]
```

Arguments

- `[-kind <kind>]` Specifies a subset of virtuals to look at. Optional. `<kind>` can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

- `[-append]` Specifies to save only virtuals that are not already saved or weren’t read in from a macro file. These unsaved virtuals are then appended to the specified or default file. Optional.

- `<filename>` Used for writing the virtual definitions. Optional. If you don’t specify `<filename>`, the default virtual filename (`virtuals.do`) will be used. You can specify a different default in the `pref.tcl` file.

See also

- `virtual count` (CR-265), "Virtual Objects (User-defined buses, and more)" (UM-161)
virtual show

The **virtual show** command lists the full path names of all explicitly defined virtuals.

Syntax

```
virtual show
[-kind <kind>]
```

Arguments

- **-kind <kind>**
  Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

See also

- **virtual define** (CR-266), **virtual describe** (CR-268), "Virtual Objects (User-defined busses, and more)" (UM-161)
**virtual signal**

The **virtual signal** command creates a new signal, known only by the GUI (not the kernel), that consists of concatenations of signals and subelements as specified in `<expressionString>`. It cannot handle bit selects and slices of Verilog registers. Please see "Syntax and conventions" (CR-9) for more details on syntax.

**Syntax**

```plaintext
virtual signal
   [-env <path>] [-install <path>] [-implicit] [-delay <time>]
   {<expressionString>}  <name>
```

**Arguments**

- `-env <path>`
  Specifies a hierarchical context for the signal names in `<expressionString>`, so they don't all have to be full paths. Optional.

- `-install <path>`
  Causes the newly-created signal to become a child of the specified region. If `-install` is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in `<expressionString>`. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region `virtuals:/Signals`. Optional.

- `-implicit`
  Used internally to create virtuals that are automatically saved with the List or Wave format. Optional.

- `-delay <time>`
  Specifies a value by which the virtual signal will be delayed. Optional. You can use negative values to look forward in time. If units are specified, the `<time>` option must be enclosed in curly braces. See the examples below for more details.

- `{<expressionString>}`
  A text string expression in the MTI GUI expression format that defines the signal and subelement concatenation. Can also be a literal constant or computed subexpression. Required. For details on syntax, please see "Syntax and conventions" (CR-9).

- `<name>`
  The name you define for the virtual signal. Required. Case is ignored unless installed in a Verilog region. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, `<name>` needs to be quoted with double quotes or with curly braces.
Examples

virtual signal -env sim:/chip/alu { (concat_range (4 downto 0))(a_04 & a_03 & a_02 & a_01 & a_00) } a
Reconstructs a bus sim:/chip/alu/a(4 downto 0), using VHDL notation, assuming that \( a_{ii} \) are scalars all of the same type.

virtual signal -env sim:chip.alu { (concat_range [4:0])&{a_04, a_03, a_02, a_01, a_00} } a
Reconstructs a bus sim:chip.alu.a[4:0], using Verilog notation. Note that the concatenation notation starts with ",&{" rather than "[".

virtual signal -install sim:/testbench { /chipa/alu/a(19 downto 13) & /chipa/decode/inst & /chipa/mode } stuff
Creates a signal sim:/testbench/stuff which is a record type with three fields corresponding to the three specified signals. The example assumes /chipa/mode is of type integer, /chipa/alu/a is of type std_logic_vector, and /chipa/decode/inst is a user-defined enumeration.

virtual signal -delay {10 ps} {/top/signalA} myDelayedSignalA
Creates a virtual signal that is the same as /top/signalA except it is delayed by 10 ps.

virtual signal { chip.instruction[23:21] } address_mode
Creates a three-bit signal, chip.address_mode, as an alias to the specified bits.

virtual signal {a & b & c & 3'b000} myextendedbus
Concatenates signals \( a, b, \) and \( c \) with the literal constant '000'.

virtual signal {num & "000"} fullbus
add wave -unsigned fullbus
Adds three missing bits to the bus num, creates a virtual signal fullbus, and then adds that signal to the wave window.

virtual signal { num31 & num30 & num29 & ... & num4 & num3 & "000" } fullbus
add wave -unsigned fullbus
Reconstructs a bus that was fragmented by synthesis and is missing the lower three bits. Note that you would have to type in the actual bit names (i.e. num28, num27, etc.) represented by the ... in the syntax above.

virtual signal {(aold == anew) & (bold == bnew)} myequalityvector
Creates a two-bit signal (with an enumerated type) based on the results of the subexpressions. For example, if aold equals anew, then the first bit is true (1). Alternatively, if bold does not equal c, the second bit is false (0). Each subexpression is evaluated independently.
### Commands fully compatible with virtual signals

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add list (CR-48)</td>
<td></td>
</tr>
<tr>
<td>add log / log (CR-166)</td>
<td></td>
</tr>
<tr>
<td>add wave (CR-57)</td>
<td></td>
</tr>
<tr>
<td>checkpoint (CR-82) and restore (CR-206)</td>
<td></td>
</tr>
<tr>
<td>delete (CR-133)</td>
<td></td>
</tr>
<tr>
<td>describe (CR-134) (&quot;virtual describe&quot; is a little faster)</td>
<td></td>
</tr>
<tr>
<td>down (CR-139) / up (CR-231)</td>
<td></td>
</tr>
<tr>
<td>examine (CR-149)</td>
<td></td>
</tr>
<tr>
<td>find (CR-153)</td>
<td></td>
</tr>
<tr>
<td>force (CR-156) / noforce (CR-173)</td>
<td></td>
</tr>
<tr>
<td>restart (CR-204)</td>
<td></td>
</tr>
<tr>
<td>left (CR-164) / right (CR-208)</td>
<td></td>
</tr>
<tr>
<td>search (CR-212)</td>
<td></td>
</tr>
<tr>
<td>searchlog (CR-214)</td>
<td></td>
</tr>
<tr>
<td>show (CR-218)</td>
<td></td>
</tr>
</tbody>
</table>

### Commands compatible with virtual signals using [virtual expand <signal>]

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>check contention add (CR-74)</td>
<td></td>
</tr>
<tr>
<td>check contention config (CR-75)</td>
<td></td>
</tr>
<tr>
<td>check contention off (CR-76)</td>
<td></td>
</tr>
<tr>
<td>check float add (CR-77)</td>
<td></td>
</tr>
<tr>
<td>check float config (CR-78)</td>
<td></td>
</tr>
<tr>
<td>check float off (CR-79)</td>
<td></td>
</tr>
<tr>
<td>check stable on (CR-81)</td>
<td></td>
</tr>
<tr>
<td>check stable off (CR-80)</td>
<td></td>
</tr>
<tr>
<td>drivers (CR-141)</td>
<td></td>
</tr>
<tr>
<td>power add (CR-184)</td>
<td></td>
</tr>
<tr>
<td>power report (CR-185)</td>
<td></td>
</tr>
<tr>
<td>power reset (CR-186)</td>
<td></td>
</tr>
<tr>
<td>toggle add (CR-225)</td>
<td></td>
</tr>
<tr>
<td>toggle reset (CR-227)</td>
<td></td>
</tr>
<tr>
<td>toggle report (CR-226)</td>
<td></td>
</tr>
<tr>
<td>vcd add (CR-233)</td>
<td></td>
</tr>
</tbody>
</table>

### Commands not currently compatible with virtual signals

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>when (CR-314)</td>
<td></td>
</tr>
</tbody>
</table>

### See also

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual count (CR-265)</td>
<td></td>
</tr>
<tr>
<td>virtual define (CR-266)</td>
<td></td>
</tr>
<tr>
<td>virtual delete (CR-267)</td>
<td></td>
</tr>
<tr>
<td>virtual describe (CR-268)</td>
<td></td>
</tr>
<tr>
<td>virtual expand (CR-269)</td>
<td></td>
</tr>
<tr>
<td>virtual function (CR-270)</td>
<td></td>
</tr>
<tr>
<td>virtual hide (CR-273)</td>
<td></td>
</tr>
<tr>
<td>virtual log (CR-274)</td>
<td></td>
</tr>
<tr>
<td>virtual nolog (CR-277)</td>
<td></td>
</tr>
<tr>
<td>virtual region (CR-279)</td>
<td></td>
</tr>
<tr>
<td>virtual save (CR-280)</td>
<td></td>
</tr>
<tr>
<td>virtual show (CR-281)</td>
<td></td>
</tr>
<tr>
<td>virtual type (CR-285)</td>
<td></td>
</tr>
<tr>
<td>Virtual Objects (User-defined buses, and more) (UM-161)</td>
<td></td>
</tr>
</tbody>
</table>
**virtual type**

The **virtual type** command creates a new enumerated type, known only by the GUI, not the kernel. Virtual types are used to convert signal values to character strings. The command works with signed integer values up to 64 bits.

**Syntax**

```
virtual type
  {<list_of_strings>}  <name>
```

**Arguments**

{<list_of_strings>}
A list of values and their associated character strings. Required. Values can be expressed in decimal or based notation. Three kinds of based notation are supported: Verilog, VHDL, and C-language styles. The values are interpreted without regard to the size of the bus to be mapped. Bus widths up to 64 bits are supported.

There is currently no restriction on the contents of each string, but if strings contain spaces they would need to be quoted, and if they contain characters treated specially by Tcl (square brackets, curly braces, backslashes...), they would need to be quoted with curly braces.

See the examples below for further syntax.

<name>
The user-defined name of the virtual type. Required. Case is not ignored. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, <name> needs to be quoted with double quotes or with curly braces.

**Examples**

```
virtual type {state0 state1 state2 state3} mystateType
virtual function {(mystateType)mysignal} myConvertedSignal
add wave myConvertedSignal
    Using positional notation, associates each string with an enumeration index, starting at zero and increasing by one in the positive direction. When myConvertedSignal is displayed in the Wave, List or Signals window, the string "state0" will appear when mysignal == 0, "state1" when mysignal == 1, "state2" when mysignal == 2, etc.

virtual type {{0 NULL_STATE} {1 st1} {2 st2} {0x04 st3} {16'h08 st4} 'h10 st5} {16#20 st6} {0b01000000 st7} {0x80 st8} {default BAD_STATE}} myMappedType
virtual function {(myMappedType)mybus} myConvertedBus
add wave myConvertedBus
    Uses sparse mapping of bus values to alphanumeric strings for an 8-bit, one-hot encoding. It shows the variety of syntax that can be used for values. The value "default" has special meaning and corresponds to any value not explicitly specified.
```
See also

virtual function (CR-270), "Virtual Objects (User-defined buses, and more)" (UM-161)

Note: Virtual types cannot be used in the when (CR-314) command.
vlib

The `vlib` command creates a design library. You must use `vlib` rather than operating system commands to create a library directory or index file. If the specified library already exists as a valid ModelSim library, the `vlib` command will exit with a warning message without touching the library.

Syntax

```
vlib [-help] [-dos | -short | -unix | -long] <directory_name>
```

Arguments

`-help`
Displays the command’s options and arguments. Optional.

`-dos`
Specifies that subdirectories in a library have names that are compatible with DOS. Not recommended if you use the `vmake` (CR-296) utility. Optional. Default for ModelSim PE.

`-short`
Interchangeable with the `-dos` argument. Optional.

`-unix`
Specifies that subdirectories in a library may have long file names that are NOT compatible with DOS. Optional. Default for ModelSim SE.

`-long`
Interchangeable with the `-unix` argument. Optional.

`<directory_name>`
Specifies the pathname of the library to be created. Required.

Examples

```
vlib design
```
Creates the design library `design`. You can define a logical name for the library using the `vmap` command (CR-297) or by adding a line to the library section of the `modelsim.ini` file that is located in the same directory.
vlog

The vlog command is used to invoke VLOG, the Model Technology Verilog compiler. Use vlog to compile Verilog source code into a specified working library (or to the work library by default).

vlog may be invoked from within ModelSim or from the operating system command prompt. It may also be invoked during simulation.

Syntax

```
```

Arguments

- **-93**
  Specifies that the VHDL interface to Verilog modules use VHDL 1076-1993 extended identifiers to preserve case in Verilog identifiers that contain uppercase letters.

- **-help**
  Displays the command’s options and arguments. Optional.

- **-compat**
  Disables optimizations that result in different event ordering than Verilog-XL. Optional. ModelSim Verilog generally duplicates Verilog-XL event ordering, but there are cases where it is inefficient to do so. Using this option does not help you find event order dependencies, but it allows you to ignore them. Keep in mind that this option does not account for all event order discrepancies, and that using this option may degrade performance. See "Event ordering in Verilog designs" (UM-92) for additional information.

- **-compile_uselibs=<directory_name>**
  Locates source files specified in a `uselib` directive (see "Verilog-XL `uselib` compiler directive" (UM-87)), compiles those files into automatically created libraries, and updates the `modelsim.ini` file with the logical mappings to the new libraries. Optional. If a `directory_name` is not specified, ModelSim uses the name specified in the `MTI_USELIB_DIR` environment variable. If that variable is not set, ModelSim creates the directory `mti_uselibs` in the current working directory.

- **-debugCellOpt**
  Produces Main window transcript output that identifies why certain cells within the design were not optimized. Used only when compiling gate-level Verilog libraries with -fast (see below). Optional.
+define+<macro_name>[=<macro_text>]  
Allows you to define a macro from the command line that is equivalent to the following compiler directive:

'define <macro_name> <macro_text>

Optional. Multiple +define options are allowed on the command line. A command line macro overrides a macro of the same name defined with the `define` compiler directive.

+delay_mode_distributed
Disables path delays in favor of distributed delays. Optional. See "Delay modes" (UM-111) for details.

+delay_mode_path
Sets distributed delays to zero in favor of using path delays. Optional. See "Delay modes" (UM-111) for details.

+delay_mode_unit
Sets path delays to zero and non-zero distributed delays to one time unit. Optional. See "Delay modes" (UM-111) for details.

+delay_mode_zero
Sets path delays and distributed delays to zero. Optional. See "Delay modes" (UM-111) for details.

-f <filename>
Specifies a file with more command line arguments. Optional. Allows complex arguments to be reused without retyping. Nesting of -f options is allowed.

-fast[=<secondary_name>] [+acc[=<spec>] [+<module>[.]]]
Increases simulation speed by allowing parameter propagation and global optimizations. Optional. To use this parameter, you must compile the source code for your entire design in a single invocation of the compiler. The following options are available:

=secondary_name
Allows you to specify a different secondary name for the optimized code. The compiler automatically assigns a secondary name to distinguish optimized code from unoptimized code that may exist in the same library. The default secondary name for optimized code is "fast"; the default secondary name for unoptimized code is "verilog".

+acc[=<spec>][+<module>[.]]
Allows you to maintain design object visibility. Note that using this option may reduce simulation speed.

<spec> is one or more of the following characters:

b–Enable access to bits of vector nets. This is necessary for PLI applications that require handles to individual bits of vector nets. Also, some user interface commands require this access if you need to operate on net bits.

c–Enable access to library cells. By default any Verilog module bracketed with a `celldefine / `endcelldefine may be optimized, and debug and PLI access may be limited. This option keeps module cell visibility.

l–Enable access to line number directives and process names.

n–Enable access to nets.

p–Enable access to ports. This disables the module inlining optimization, and is necessary only if you have PLI applications that require access to port handles.
r–Enable access to registers (including memories, integer, time, and real types).

If <spec> is omitted, access is enabled for all objects.

<module> is a module name, optionally followed by "." to indicate all children of the module. Multiple modules are allowed, with each separated by a "+". If no modules are specified, then all modules are affected.

Note: Please see additional discussion about -fast in "Compiling for faster performance" (UM-99). Also, see +opt argument below.

-forcecode
Forces code generation for optimized inline modules when using the -fast switch.
Optional. Use only in conjunction with the -fast switch. By default, code is not generated for inline modules when the -fast switch is used.

-hazards
Detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes. Optional. You must also specify this argument when you simulate the design with vsim (CR-298). See “Hazard detection” (UM-95) for more details.

+incdir+<directory>
Specifies directories to search for files included with 'include compiler directives.
Optional. By default, the current directory is searched first and then the directories specified by the +incdir options in the order they appear on the command line. You may specify multiple +incdir options as well as multiple directories separated by "+" in a single +incdir option.

-incr
Performs an incremental compile. Optional. Compiles only code that has changed, or if compile options change.

-keep_delta
Disables optimizations that remove delta delays. Optional.
Delta delays result from zero delay events. Those events are normally processed in the next iteration or "delta" of the current timestep. -fast and +opt implement optimizations that can remove delta delays and process an event earlier.

-L <libname>
Searches the specified resource library for precompiled modules. Optional.
This argument can be used in tandem with -fast (see above) when you need to optimize pre-compiled modules for which you don’t have source code. The library search options you specify here must also be specified when you run the vsim command (CR-298).

-Lf <libname>
Same as -L, but the specified library is searched before any ’uselib directives. (See "Library usage" (UM-85) and "Verilog-XL ’uselib compiler directive" (UM-87) for more information). Optional.

-libext+<suffix>
Works in conjunction with the -y option. Specifies file extensions for the files in a source library directory. Optional. By default the compiler searches for files without extensions. If you specify the +libext option, then the compiler will search for a file with the suffix
appended to an unresolved name. You may specify only one `+libext` option, but it may contain multiple suffixes separated by "+". The extensions are tried in the order they appear in the `+libext` option.

`+librescan`
Scans libraries in command-line order for all unresolved modules. Optional.

`-line <number>`
Starts the compiler on the specified line in the Verilog source file. Optional. By default, the compiler starts at the beginning of the file.

`-lint`
Instructs ModelSim to perform three lint-style checks: 1) warn when Module ports are NULL; 2) warn when assigning to an input port; 3) warn when referencing undeclared variables/nets in an instantiation. The warnings are reported as WARNING[8]. Can also be enabled using the `Show_Lint` variable in the `modelsim.ini` file.

`+maxdelays`
Selects maximum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

`+mindelays`
Selects minimum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

`+nocheckALL`
Enables all `+nocheck` arguments described below. Optional. Argument has an effect only when compiling gate-level cell libraries with `-fast` (see above). The `+nocheck` switches increase the optimizations of `-fast`.

`+nocheckCLUP`
Allows connectivity loops in a cell to be optimized. Optional. Argument has an effect only when compiling gate-level cell libraries with `-fast` (see above).

`+nocheckDNET`
Allows both the port and the delayed port (created for negative setup/hold) to be used in the functional section of the cell. Optional. Argument has an effect only when compiling gate-level cell libraries with `-fast` (see above).

`+nocheckOPRD`
Allows an output port to be read internally by the cell. Optional. Argument has an effect only when compiling gate-level cell libraries with `-fast` (see above). Note that if the value read is the only value contributed to the output by the cell, and if there’s a driver on the net outside the cell, the value read will not reflect the resolved value.

`+nocheckSUDP`
Allows a sequential UDP to drive another sequential UDP. Optional. Argument has an effect only when compiling gate-level cell libraries with `-fast` (see above).

`-nodebug[=ports | =pli]`
Hides the internal data of the compiled design unit. Optional. The design unit’s source code, internal structure, registers, nets, etc. will not display in ModelSim’s windows. In addition, none of the hidden objects may be accessed through the Dataflow window or
with commands. This also means that you cannot set breakpoints or single step within this code. Don’t compile with this switch until you’re done debugging.

Note that this is not a speed switch like the “nodebug” option on many other products. Use the -fast switch to increase simulation speed.

The optional =ports switch hides the ports for the lower levels of your design; it should be used only to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

The optional =pli switch prevents the use of pli functions to interrogate individual modules for information; this switch may be used at any level of the design. Combine both switches with =ports+pli or =pli+ports.

**Note: -nodebug** provides protection for proprietary model information. The Verilog ‘protect’ compiler directive provides similar protection, but uses a Cadence encryption algorithm that is unavailable to Model Technology.

See additional discussion in "Source code security and -nodebug" (UM-492).

-noincr
Disables incremental compile previously turned on with -incr. Optional.

+nolibcell
By default all modules compiled from a source library are treated as though they contain a ‘celldesign’ compiler directive. This option disables this default. The ‘celldesign’ directive only affects the PLI access routines acc_next_cell and acc_next_cell_load. Optional.

-nologo
Disables the startup banner. Optional.

+nospecify
Disables specify path delays and timing checks. Optional.

+notimingchecks
Disables all timing check system tasks completely. Optional. Note that there is no way to disable timing checks on specific instances—it’s all or nothing.

+nowarn<CODE>
Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example,

** WARNING: (vsim-3017) test.v(2): [TFMPC] - Too few port connections.
Expected <m>, found <n>.

This warning message can be disabled with the +nowarnTFMPC option.

-O0 | -O1 | -O4 | -O5
Lower the optimization to a minimum with -O0 (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Enable PE-level optimization with -O1. Optional.

Enable maximum optimization with `-O5`. Optional. Use caution with this switch. We recommend use of this switch with large sequential blocks only; other uses may significantly increase compile times. Also, before using `-O5` with `-fast` (described above), try using both switches independently to make sure the optimized design behaves the same as the original version.

`+opt+[<lib>.]<module>`

Generates optimized code for designs that have been previously compiled unoptimized (without the `-fast` option; see above). Optional. The `<module>` specification is the name of the top-level design module, and `<lib>`, which is optional, is the library in which it resides. By default, the top-level module is searched for in the work library. If the design has multiple top-level modules, then provide the names in a list separated by plus signs. For example,

```
vlog +opt+testbench+globals
```

Any options that are appropriate with `-fast` are also appropriate with `+opt`. Specifically, use the `+acc` option to enable PLI access, and use the `-L` and `-Lf` options to specify the libraries to be searched.

**Note:** Please see additional discussion about `+opt` and `-fast` in "Compiling for faster performance" (UM-99).

`-quiet`

Disables 'loading' messages. Optional.

`-R [<simargs>]`

Instructs the compiler to invoke the simulator (`vsi`m (CR-298)) after compiling the design. The compiler automatically determines which top-level modules are to be simulated. The command line arguments following `-R` are passed to the simulator, not the compiler. Place the `-R` option at the end of the command line or terminate the simulator command line arguments with a single "-" character to differentiate them from compiler command line arguments.

The `-R` option is not a Verilog-XL option, but it is used by ModelSim to combine the compile and simulate phases together as you may be used to doing with Verilog-XL. It is not recommended that you regularly use this option because you will incur the unnecessary overhead of compiling your design for each simulation run. Mainly, it is provided to ease the transition to ModelSim.

`-refresh`

Regenerates a library image. Optional. By default, the work library is updated; use `-work <library_name>` to update a different library. See vlog examples for more information.

`-source`

Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.

`-time`

Reports the "wall clock time" vlog takes to compile the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on vlog.
+typdelays
Selects typical delays from the "min:typ:max" expressions. Default. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

-u
Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names. Optional.

-v <library_file>
Specifies a source library file containing module and UDP definitions. Optional. See "Verilog-XL compatible compiler arguments" (UM-86) for more information.

After all explicit filenames on the vlog command line have been processed, the compiler uses the -v option to find and compile any modules that were referenced but not yet defined. Modules and UDPs within the file are compiled only if they match previously unresolved references. Multiple -v options are allowed. See additional discussion in the examples.

-version
Returns the version of the compiler as used by the licensing tools, such as "Model Technology ModelSim SE vlog 5.5 Compiler 2000.01 Jan 28 2000".

-work <library_name>
Specifies a logical name or pathname of a library that is to be mapped to the logical library work. Optional; by default, the compiled design units are added to the work library. The specified pathname overrides the pathname specified for work in the project file.

-y <library_directory>
Specifies a source library directory containing module and UDP definitions. Optional. See "Verilog-XL compatible compiler arguments" (UM-86) for more information.

After all explicit filenames on the vlog command line have been processed, the compiler uses the -y option to find and compile any modules that were referenced but not yet defined. Files within this directory are compiled only if the file names match the names of previously unresolved references. Multiple -y options are allowed. You will need to specify a file suffix by using -y in conjunction with the +libext+<suffix> option if your filenames differ from your module names. See additional discussion in the examples.

Specifying the name of the Verilog source code file to compile. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used.

Examples

vlog example.vlg
Compiles the Verilog source code contained in the file example.vlg.

vlog -nodebug example.v
Hides the internal data of example.v. Models compiled with -nodebug cannot use any of the ModelSim debugging features; any subsequent user will not be able to see into the model.

vlog -nodebug=ports level3.v level2.v
vlog -nodebug top.v
The first line compiles and hides the internal data, plus the ports, of the lower-level design units, level3.v and level2.v. The second line compiles the top-level unit, top.v,
without hiding the ports. It is important to compile the top level without =ports because top-level ports must be visible for simulation.

vlog -nodebug=ports+pli level3.v level2.v
vlog -nodebug=pli top.v

The first command hides the internal data, and ports of the design units, level3.v and level2.v. In addition it prevents the use of PLI functions to interrogate the compiled modules for information (either =ports+pli or =pli+ports works fine for this command). The second line compiles the top-level unit without hiding the ports but restricts the use of PLI functions as well.

Note that the =pli switch may be used at any level of the design but =ports should only be used on lower levels since you can’t simulate without visible top-level ports.

See "Source code security and -nodebug" (UM-492) for more details.

vlog -fast cpu_rtl.v
Compiles all modules in cpu_rtl.v using global optimizations. Assuming your top-level module is named testbench, you would simulate the design as follows:

vsim -c testbench

vlog -fast=opt1 cpu_rtl.v
Compiles all modules in cpu_rtl.v using global optimizations, and assigns the secondary name "opt1" to the optimized modules.

vlog -fast +acc=rn cpu_rtl.v
Compiles cpu_rtl.v using global optimizations, but enables net and register access in all modules in the design.

Note: Please see additional discussion about -fast in "Compiling for faster performance" (UM-99) in the Verilog simulation chapter.

vlog top.v -v und1
After compiling top.v, vlog will scan the file und1 for modules or primitives referenced but undefined in top.v. Only referenced definitions will be compiled.

vlog top.v +libext+.v+.u -y vlog_lib
After compiling top.v, vlog will scan the vlog_lib library for files with modules with the same name as primitives referenced, but undefined in top.v. The use of +libext+.v+.u implies filenames with a .v or .u suffix (any combination of suffixes may be used). Only referenced definitions will be compiled.

vlog -work mylib -refresh
The -work option specifies mylib as the library to regenerate. -refresh rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim (4.6 and later only).

If your library contains VHDL design units be sure to regenerate the library with the vcom command (CR-252) using the -refresh option as well. See "Regenerating your design libraries" (UM-55) for more information.

vlog module1.v -u -O0 -incr
The -incr option determines whether or not the module source or compile options have changed as module1 is parsed. If no change is found, the code generation phase is skipped. Differences in compile options are determined by comparing the compiler options stored in the _info file with the compiler options given. They must match exactly.
vmake

The vmake utility allows you to use a UNIX or Windows MAKE program to maintain libraries. The vmake utility is run on a compiled design library, and outputs a makefile that can be used to reconstruct the library. The resulting makefile can then be run with a version of MAKE (not supplied with ModelSim); a MAKE program is included with Microsoft Visual C/C++, as well as many other program development environments.

After running the vmake utility, MAKE will recompile only the design units (and their dependencies) that have changed. Vmake needs to be run only once, then you can simply run MAKE to rebuild your design. If you add new design units or delete old ones, you should re-run vmake to generate a new makefile.

This command must be invoked from either the UNIX or the Windows/DOS prompt.

Syntax

vmake
   [-fullsrcpath] [-help] [<!--library_name>] [--><<makefile>]

Arguments

-fullsrcpath
   Produces complete source file paths within generated makefiles. Optional. By default source file paths are relative to the directory in which compiles originally occurred. This argument makes it possible to copy and evaluate generated makefiles within directories that are different from where compiles originally occurred.

-help
   Displays the command’s options and arguments. Optional.

<library_name>
   Specifies the library name; if none is specified, then work is assumed. Optional.

>>makefile
   Specifies the makefile name. Optional.

Examples

To produce a makefile for the work library:
   vmake >makefile

You can also run vmake on libraries other than work:
   vmake mylib >mylib.mak

To rebuild mylib, specify its makefile when you run MAKE:
   make -f mylib.mak
vmap

The vmap command defines a mapping between a logical library name and a directory by modifying the modelsim.ini file. With no arguments, vmap reads the appropriate modelsim.ini file(s) and prints the current logical library to physical directory mappings. Returns nothing.

Syntax

vmap
   [ -help ] [ -c ] [ -del ] [ <logical_name> ] [ <path> ]

Arguments

- help
  Displays the command’s options and arguments. Optional.

- c
  Copies the default modelsim.ini file from the ModelSim installation directory to the current directory. Optional.

- del
  Deletes the mapping specified by <logical_name> from the current project file. Optional.

<logical_name>
  Specifies the logical name of the library to be mapped. Optional.

<path>
  Specifies the pathname of the directory to which the library is to be mapped. Optional. If omitted, the command displays the mapping of the specified logical name.
vsim

The **vsim** command is used to invoke the VSIM simulator, or to view the results of a previous simulation run (when invoked with the `-view` switch). You can specify a configuration, an entity/architecture pair, or a module for simulation. If a configuration is specified, it is invalid to specify an architecture. With no options, **vsim** brings up the Load Design dialog box, allowing you to specify the design and options; the Load Design dialog box will not be presented if you specify any options. During elaboration **vsim** determines if the source has been modified since the last compile.

To **manually interrupt design elaboration** use the Break key or <control-c> from a shell.

The **vsim** command may also be invoked from the command line within ModelSim with most of the options shown below (all except the **vsim** -c and -restore options).

**Syntax**

```
vsim
    [-assertfile <filename>] [-c] [-compress_elab] [-coverage]
    [-do "<command_string>" | <macro_file_name>] [+dumpports+direction]
    [-elab <filename>] [-elab_cont <filename>] [-elab_defer_fli]
    [-f <filename>] [-filemap_elab <HDLfilename>=<NEWfilename>]
    [-g<Name>=<Value> ...] [-G<Name>=<Value> ...] [-gui]
    [-HELP] [-i] [-installcolormap] [-keeploaded] [-keeploadedrestart]
    [-keepstdout] [-l <filename>] [-license_option] [-load_elab <filename>]
    [-multisource_delay min | max | latest] [+multisource_int_delays]
    [-nocompress] [+no_notifier] [+no_tchk_msg] [+notimingchecks] [-quiet]
    [-restore <filename>]
    [-sdfmin | -sdftyp | -sdffmax [instancename]=<sdf_filename>]
    [-sdfnerror] [-sdfnowarn] [+sdf_verbose] [-t <multiplier>|<time_unit>]
    [-tag <string>] [-title <title>] [-trace_foreign <int>] [-vcdatim]
    [-version] [-view [<dataset_name>=]<WLF_filename> [-wlf <filename>]
    [-wlfcompres] [-wlfnocompress] [-wlfslim <size>] [-wlftlim <duration>]
    [-absentisempty] [-foreign <attribute>] [-nocollapse] [-nofileshare]
    [-noglitch] [+no_glitch_msg] [-std_input <filename>]
    [-std_output <filename>] [-strictvital] [-vcdread <filename>]
    [-vital2.2b]
    [+alt_path_delays] [-extend_tcheck_data_limit <percent>]
    [-extend_tcheck_ref_limit <percent>]
    [-hazards] [+int_delays] [-L <library_name> ...] [-Lf <library_name> ...]
    [+maxdelays] [+mindelays] [+nocancelled_e_msg] [+no_neg_tchk]
    [+no notifier] [+no_path_edge] [+no_pulse_msg] [+nshow_cancelled_e]
    [+nosdferror] [+nosdfwarn] [+nospecify] [+no_tchk_msg] [+nowarn<CODE>]
    [-ntc_warn] [-pli "<object list>"] [+<plusarg>]
    [+pulse_e/<percent>] [+pulse_e_style_ondetect] [+pulse_e_style_onevent]
    [+pulse_int_e/<percent>] [+pulse_int_r/<percent>] [+pulse_r/<percent>]
    [+sdf_nocheck_celltype] [+show_cancelled_e] [+transport_int_delays]
    [+transport_path_delays] [+typdelays]
    [-v2k_int_delays]
    [-<library_name>..<design_unit>]
```

VSIM arguments are grouped alphabetically by language:

- Arguments, VHDL and Verilog (CR-299)
- Arguments, VHDL (CR-305)
Arguments, VHDL and Verilog

-assertfile <filename>
  Designates an alternative file for recording assertion messages. Optional. By default assertion messages are output to the file specified by the TranscriptFile variable in the modelsim.ini file (see "Creating a transcript file" (UM-452)).

-c
  Specifies that the simulator is to be run in command line mode. Optional. Also see "Running command-line and batch-mode simulations" (UM-490) for more information.

-compress_elab
  Compresses an elaboration file when it is created. Optional. See "Simulating with an elaboration file" (UM-108) for more information.

-coverage
  Allows line number execution statistics to be kept by the simulator. By default no statistics are kept. This switch must be specified during command-line invocation of the simulator in order to use the various coverage commands: coverage clear (CR-116), coverage reload (CR-121), and coverage report (CR-122). Also see Chapter 10 - Code Coverage for more information. Optional.

-do "<command_string>" | <macro_file_name>
  Instructs VSIM to use the command(s) specified by <command_string> or the macro file named by <macro_file_name> rather than the startup file specified in the .ini file, if any. Optional.

+dumpports+direction
  Modifies the format of extended VCD files to contain direction information.

-elab <filename>
  Creates an elaboration file for use with -load_elab. Optional. See "Simulating with an elaboration file" (UM-108) for more information.

-elab_cont <filename>
  Creates an elaboration file for use with -load_elab and then continues the simulation. Optional.

-elab_defer_fli
  Defers the initialization of FLI models until the load of the elaboration file. Use this argument along with -elab to create elaboration files for designs with FLI models that don’t support checkpoint/restore. Note that FLI models sensitive to design load ordering may still not work correctly even if you use this argument. See "Simulating with an elaboration file" (UM-108) for more information.

-f <filename>
  Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Optional.

-filemap_elab <HDLfilename>=<NEWfilename>
  Defines a file mapping during -load_elab that lets you change the stimulus. Optional. See "Simulating with an elaboration file" (UM-108) for more information.
-g>Name=<Value>  
Assigns a value to all specified VHDL generics and Verilog parameters that have not  
received explicit values in generic maps, instantiations, or via defparams (such as top-  
level generics/parameters and generics/parameters that would otherwise receive their  
default values). Optional. Note there is no space between -g and <Name>=<Value>.  

Name is the name of the generic/parameter, exactly as it appears in the VHDL source  
(case is ignored). Value is an appropriate value for the declared data type of a VHDL  
generic or any legal value for a Verilog parameter. Make sure the Value you specify for 
a VHDL generic is appropriate for VHDL declared data types. VHDL type mismatches  
will cause the specification to be ignored (including no error messages).  

No spaces are allowed anywhere in the specification, except within quotes when  
specifying a string value. Multiple -g options are allowed, one for each generic/  
parameter.  

Name may be prefixed with a relative or absolute hierarchical path to select generics in  
an instance-specific manner. For example,  

Specifying -g/top/u1/tpd=20ns on the command line would affect only the tpd generic  
on the /top/u1 instance, assigning it a value of 20ns.  

Specifying -gu1/tpd=20ns affects the tpd generic on all instances named u1.  

Specifying -gtpd=20ns affects all generics named tpd.  

If more than one -g option selects a given generic the most explicit specification takes  
precedence. For example,  

```
vsim -g/top/ram/u1/tpd_hl=10ns -gtpd_hl=15ns top
```

This command sets tpd_hl to 10ns for the /top/ram/u1 instance. However, all other tpd_hl  
generics on other instances will be set to 15ns.  

Limitation: In general, generics/parameters of composite type (arrays and records)  
cannot be set from the command line. However, you can set string arrays, std_logic  
vectors, and bit vectors if they can be set using a quoted string. For example,  

```
-gstrgen="This is a string"
-gslv="01001110"
```

The quotation marks must make it into vsim as part of the string because the type of the  
value must be determinable outside of any context. Therefore, when entering this  
command from a shell, put a forward tick around the string. For example:  

```
-gstrgen="This is a string"
```

If working within the ModelSim GUI, you would enter the command as follows:  

```
{-gstrgen="This is a string"}
```

▶ **Note:** When you compile Verilog code with -fast (see vlog (CR-288)), all parameter  
values are set at compile time. Therefore, the -g option has no effect on these parameters.  

-G>Name=<Value>  
Same as -g (see above) except that it will also override generics/parameters that received  
explicit values in generic maps, instantiations, or via defparams. Optional. Note there is  
no space between -G and <Name>=<Value>.
-gui
Starts the ModelSim GUI without loading a design. Optional.

-help
Displays the command’s options and arguments. Optional.

-i
Specifies that the simulator is to be run in interactive mode. Optional.

-installcolormap
For UNIX only. Causes vsim to use its own colormap so as not to hog all the colors on the display. This is similar to the -install switch on Netscape. Optional.

-keeploaded
Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries when it restarts or loads a new design. Optional. The shared libraries will remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart in order for this to work effectively.

-keeploadedrestart
Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries during a restart. Optional. The shared libraries will remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart in order for this to work effectively.

We recommend using this option if you’ll be doing warm restores after a restart and the user application code has set callbacks in the simulator. Otherwise, the callback function pointers might not be valid if the shared library is loaded into a new position.

-keepstdout
For use with foreign programs. Instructs the simulator to not redirect the stdout stream to the Main window. Optional.

-l <filename>
Saves the contents of the “Main window” (UM-173) transcript to <filename>. Optional. Default is transcript. Can also be specified using the .ini (see "Creating a transcript file" (UM-452)) file or the.tcl preference file.

<license_option>
Restricts the search of the license manager. Optional. Use one of the following options.

<table>
<thead>
<tr>
<th>&lt;license_option&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lic_nomgc</td>
<td>exclude any MGC licenses from the search</td>
</tr>
<tr>
<td>-lic_nomti</td>
<td>exclude any MTI licenses from the search</td>
</tr>
<tr>
<td>-lic_noqueue</td>
<td>do not wait in queue when license is unavailable</td>
</tr>
<tr>
<td>-lic_plus</td>
<td>checks out ModelSim SE/PLUS (VHDL and Verilog) license immediately after invocation</td>
</tr>
<tr>
<td>-lic_vhdl</td>
<td>checks out ModelSim SE/VHDL license immediately after invocation</td>
</tr>
<tr>
<td>-lic_vlog</td>
<td>checks out ModelSim SE/VLOG license immediately after invocation</td>
</tr>
</tbody>
</table>
The options may also be specified with the License (UM-448) variable in the modelsim.ini file. Note that settings made from the command line are additive to options set in the License variable. For example, if you set the License variable to nomgc and use the -lic_plus option from the command line, vsim will check out only MTI SE/PLUS licenses.

-**load_elab** <filename>
  Loads an elaboration file that was created with **elab**. Optional. See "Simulating with an elaboration file" (UM-108) for more information.

-**multisource_delay** min | max | latest
  Controls the handling of multiple PORT or INTERCONNECT constructs that terminate at the same port. Optional. By default, the Module Input Port Delay (MIPD) is set to the max value encountered in the SDF file. Alternatively, you may choose the min or latest of the values. If you have a Verilog design and want to model multiple interconnect paths independently, use the +multisource_int_delays switch (see "Arguments, Verilog" (CR-306)).

  +**multisource_int_delays**
  Enables multisource interconnect delay with pulse handling and transport delay behavior. Optional. Use this argument when you have interconnect data in your SDF file and you want the delay on each interconnect path modeled independently. Pulse handling is configured using the +pulse_int_e and +pulse_int_r switches (described below).

-**nocompress**
  Causes VSIM to create uncompressed checkpoint files. Optional. This option must be used with the -restore option (below) to restore a simulation from an uncompressed checkpoint file. This option may also be specified with the CheckpointCompressMode (UM-447) variable in the modelsim.ini file.

  +**no_notifier**
  Disables the toggling of the notifier register argument of the timing check system tasks. Optional. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation on timing violations.

  +**no_tchk_msg**
  Disables error messages issued by timing check system tasks when timing check violations occur. Optional. Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

  +**notimingchecks**
  Disables Verilog and VITAL timing checks for faster simulation. Optional. By default, Verilog timing check system tasks ($setup, $hold,...) in specify blocks are enabled. For VITAL, the timing check default is controlled by the ASIC or FPGA vendor, but most default to enabled.

-**quiet**
  Disable 'loading' messages during batch-mode simulation. Optional.

-**restore** <filename>
  Specifies that VSIM is to restore a simulation saved with the checkpoint command (CR-82). Optional. Use the -nocompress switch (above) if compression was turned off when the checkpoint command (CR-82) was used or if VSIM was initially invoked with
-nocompress. See additional discussion in "How to use checkpoint/restore" (UM-488); -nocompress is also an option of the restore command (CR-206).

**Note:** You must restore vsim under the same environment in which you did the checkpoint. This means not only the same type of machine and OS and at least the same memory size, but also the same vsim environment such as GUI vs. command line mode.

-sdfmin | -sdfmax | -sdfmax [{instance}=<sdf_filename>]
Annotates VITAL or Verilog cells in the specified SDF file (a Standard Delay Format file) with minimum, typical, or maximum timing. Optional. The use of [{instance}=<sdf_filename>] with <sdf_filename> is also optional; it is used when the backannotation is not being done at the top level. See "Specifying SDF files for simulation" (UM-378).

-sdfnoerror
Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

-sdfnowarn
Disables warnings from the SDF reader. Optional. See Chapter 4 - VHDL simulation for an additional discussion of SDF.

+sdf_verbose
Turns on the verbose mode during SDF annotation. The Main window provides detailed warnings and summaries of the current annotation. Optional.

-t [{multiplier}][<time_unit>]
Specifies the simulator time resolution. Optional. <time_unit> must be one of the following:

*fs, ps, ns, us, ms, sec*

The default is 1ns; the optional <multiplier> may be 1, 10 or 100. Note that there is no space between the multiplier and the unit (i.e., 10fs, not 10 fs).

If you omit the -t argument, the default time resolution depends on design type: in a Verilog design with "timescale" directives, the minimum time precision is used (see "Simulator resolution limit" (UM-90) for further details); in Verilog designs without any timescale directives, or in a VHDL or mixed design, the value specified for the Resolution (UM-449) variable in the modelsim.ini file is used.

Once you've begun simulation, you can determine the current simulator resolution by invoking the report command (CR-202) with the simulator state option.

-tag <string>
Specifies a string tag to append to foreign trace filenames. Optional; used with the -trace_foreign <int> option. Used when running multiple traces in the same directory. See "Invoking a trace" (FLI-37).

-title <title>
Specifies the title to appear for the ModelSim Main window. Optional. If omitted the current ModelSim version is the window title. Useful when running multiple simultaneous simulations. Text strings with spaces must be in quotes (e.g., "my title").
-trace_foreign <int>
Creates two kinds of foreign interface traces: a log of what functions were called, with
the value of the arguments, and the results returned; and a set of C-language files to
replay what the foreign interface side did.

The purpose of the logfile is to aid the debugging of your FLI/PLI/VPI code. The primary
purpose of the replay facility is to send the replay file to MTI support for debugging co-
simulation problems, or debugging problems for which it is impractical to send the FLI/
PLI/VPI code. See “Invoking a trace” (FLI-37) for more information.

-vcdstim <filename>
Resimulates a design from a VCD file. Optional. The VCD file must have been created
in a previous simulation using the vcd dumpports command (CR-236). See
"Resimulating a design from a VCD file" (UM-395) for more information.

-version
Returns the version of the simulator as used by the licensing tools, such as "Model
Technology ModelSim SE vsim 5.5 Simulator 2000.01 Jan 28 2000".

-view [<dataset_name>=]<WLF_filename>
Specifies a wave log format (WLF) file for vsim to read. Allows you to use VSIM to view
the results from an earlier simulation. The Structure, Signals, Wave, and List windows
can be opened to look at the results stored in the WLF file (other ModelSim windows will
not show any information when you are viewing a dataset). See additional discussion in
"Examples” (CR-311).

-wlf <filename>
Specifies the name of the wave log format (WLF) file to create. The default is vsim.wlf.
Optional.

-wlfcompress
Creates compressed WLF files. Default. Use -wlfnocompress to turn off compression.

-wlfnocompress
Causes VSIM to create uncompressed WLF files. Optional. Beginning with version 5.5,
WLF files are compressed by default in order to reduce file size. This may slow
simulation speed by one to two percent. You may want to disable compression to speed
up simulation or if you are experiencing problems with faulty data in the resulting WLF
file. This option may also be specified with the WLFCompress (UM-450) variable in the
modelsim.ini file.

-wlfslim <size>
Specifies a size restriction in megabytes for the event portion of the WLF file. Optional.
The default is infinite size (0). The <size> must be an integer.

Note that a WLF file contains event, header, and symbol portions. The size restriction is
placed on the event portion only. When ModelSim exits, the entire header and symbol
portion of the WLF file is written. Consequently, the resulting file will be larger than the
size specified with -wlfslim.

If used in conjunction with -wlflim, the more restrictive of the limits will take effect.
This option may also be specified with the WLFSizeLimit (UM-450) variable in the
modelsim.ini file.
-wlftlim <duration>
Specifies the duration of simulation time for WLF file recording. Optional. The default is infinite time (0). The <duration> is an integer of simulation time at the current resolution; you can optionally specify the resolution if you place curly braces around the specification. For example,

{5000 ns}

sets the duration at nanoseconds regardless of the current simulator resolution.

The time range begins at current simulation time and moves back in simulation time for the specified duration. For example,

vsim -wlftlim 5000

writes at least the last 5000ns of the current simulation to the WLF file (the current simulation resolution in this case is ns).

If used in conjunction with -wlfslim, the more restrictive of the limits will take effect.

This option may also be specified with the WLFTimeLimit (UM-450) variable in the modelsim.ini file.

Note: The -wlfslim and -wlftlim switches were designed to help users limit WLF file sizes for long or heavily logged simulations. When small values are used for these switches, the values may be overridden by the internal granularity limits of the WLF file format.

Arguments, VHDL

-absentisempty
Causes VHDL files opened for read that target non-existent files to be treated as empty, rather than ModelSim issuing fatal error messages. Optional.

-foreign <attribute>
Specifies the foreign module to load. Optional. <attribute> is a quoted string consisting of the name of a C function and a path to a shared library. For example,

vsim -foreign "c_init for.sl"

You can load up to ten foreign modules. Syntax for the attribute is further described in the Introduction chapter of the ModelSim FLI Reference.

-nocollapse
Disables the optimization of internal port map connections. Optional.

-nofileshare
By default ModelSim shares a file descriptor for all VHDL files opened for write or append that have identical names. The -nofileshare switch turns off file descriptor sharing. Optional.

-noglitch
Disables VITAL glitch generation. Optional.

See Chapter 4 - VHDL simulation for additional discussion of VITAL.

-no_glitch_msg
Disable VITAL glitch error messages. Optional.
-std_input <filename>
Specifies the file to use for the VHDL TextIO STD_INPUT file. Optional.

-std_output <filename>
Specifies the file to use for the VHDL TextIO STD_OUTPUT file. Optional.

-strictvital
Exactly match the VITAL package ordering for messages and delta cycles. Optional. Useful for eliminating delta cycle differences caused by optimizations not addressed in the VITAL LRM. Using this argument negatively impacts simulator performance.

-vcdread <filename>
Simulates the VHDL top-level design from the specified VCD file. Optional. This argument is included for backwards compatibility. Resimulations in ModelSim versions 5.5c and newer should use the -vcdstim argument. See "Resimulating a design from a VCD file" (UM-395) for more details.

-vital2.2b
Selects SDF mapping for VITAL 2.2b (default is VITAL 2000). Optional.

Arguments, Verilog

+alt_path_delays
Configures path delays to operate in inertial mode by default. Optional. In inertial mode, a pending output transition is cancelled when a new output transition is scheduled. The result is that an output may have no more than one pending transition at a time, and that pulses narrower than the delay are filtered. The delay is selected based on the transition from the cancelled pending value of the net to the new pending value. The +alt_path_delays option modifies the inertial mode such that a delay is based on a transition from the current output value rather than the cancelled pending value of the net. This option has no effect in transport mode (see +pulse_e/<percent> and +pulse_r/<percent>).

-extend_tcheck_data_limit <percent>
-extend_tcheck_ref_limit <percent>
Causes a one-time extension of qualifying data or reference limits in an attempt to provide a delay net delay solution prior to any limit zeroing. Optional. (See "Negative timing check limits" (UM-96) for related information.)

<percent> is the maximum percent of limit relaxation. A limit qualifies if it bounds a violation region which does not overlap a related violation region.

For example,

$setuphold( posedge clk, posedge d, 45, 70, notifier,,,dclk,dd);
$setuphold( posedge clk, negedge d, 216, -68, notifier,,,dclk,dd);

are the same check type and have the same delay nets and thus are related.

The delay net delay analysis in this case does not provide a solution. The required negative hold delay of 68 between d and dd could cause a non-violating posedge d transition to be delayed on dd so that it could arrive after dclk for functional evaluation. By default the -68 hold limit is set pessimistically to 0 to insure the correct functional evaluation. Alternatively, you could use -extend_tcheck_data_limit so the regions overlap. This can be accomplished by extending the 216, -68 region to 216, -44. You would set -extend_tcheck_data_limit to 16 (216-68 = 148 * .16 = 24).
-hazards
Enables event order hazard checking in Verilog modules. Optional. You must also specify this argument when you compile your design with \texttt{vlog} (CR-288). See "Hazard detection" (UM-95) for more details.

+int_delays
Optimizes annotation of interconnect delays for designs that have been compiled using \texttt{-fast} (see \texttt{vlog} command (CR-288)). Optional. This argument causes \texttt{vsim} to insert "placeholder" delay elements at optimized cell inputs, resulting in faster backannotation of interconnect delay from an SDF file.

-L <library_name> ...
Specifies the library to search for design units instantiated from Verilog. See "Library usage" (UM-85) for more information. If multiple libraries are specified, each must be preceded by the \texttt{-L} option.

-Lf <library_name> ...
Same as \texttt{-L} but libraries are searched before 'uselib directives. See "Library usage" (UM-85) for more information. Optional.

+maxdelays
Selects the maximum value in min:typ:max expressions. Optional. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.

+mindelays
Selects the minimum value in min:typ:max expressions. Optional. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.

+no_cancelled_e_msg
Disables negative pulse warning messages. Optional. By default \texttt{vsim} issues a warning and then filters negative pulses on specify path delays. You can drive an X for a negative pulse using \texttt{+show_cancelled_e}.

+no_neg_tchk
Disables negative timing check limits by setting them to zero. Optional. By default negative timing check limits are enabled. This is just the opposite of Verilog-XL, where negative timing check limits are disabled by default, and they are enabled with the \texttt{+neg_tchk} option.

+no_notifier
Disables the toggling of the notifier register argument of the timing check system tasks. Optional. By default the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation on timing violations.

+no_path_edge
Causes ModelSim to ignore the input edge specified in a path delay. Optional. The result of this argument is that all edges on the input are considered when selecting the output delay. Verilog-XL always ignores the input edges on path delays.

+no_pulse_msg
Disables the warning message for specify path pulse errors. Optional. A path pulse error occurs when a pulse propagated through a path delay falls between the pulse rejection limit and pulse error limit set with the \texttt{+pulse_r} and \texttt{+pulse_e} options. A path pulse error results in a warning message, and the pulse is propagated as an X. The \texttt{+no_pulse_msg} option disables the warning message, but the X is still propagated.
+no_show_cancelled_e
Filters negative pulses on specify path delays so they don’t show on the output. Default. Use +show_cancelled_e to drive a pulse error state.

+no_tchk_msg
Disables error messages issued by timing check system tasks when timing check violations occur. Optional.Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

+nosderror
Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

+nosdwarn
Disables warnings from the SDF annotator. Optional.

+nospecify
Disables specify path delays and timing checks. Optional.

+nowarn<CODE>
Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example:

** WARNING: (vsim-3017) test.v(2): [TFMPC] - Too few port connections. Expected <m>, found <n>.

This warning message can be disabled with +nowarnTFMPC.

+ntc_warn
Enables warning messages from the negative timing constraint algorithm. Optional. By default, these warnings are disabled.

This algorithm attempts to find a set of delays for the timing check delayed net arguments such that all negative limits can be converted to non-negative limits with respect to the delayed nets. If there is no solution for this set of limits, then the algorithm sets one of the negative limits to zero and recalculates the delays. This process is repeated until a solution is found. A warning message is issued for each negative limit set to zero.

-pli "<object_list>"
Loads a space-separated list of PLI shared objects. Optional. The list must be quoted if it contains more than one object. This is an alternative to specifying PLI objects in the Veriuser entry in the modelsim.ini file, see "Preference variables located in INI files" (UM-444). You can use environment variables as part of the path.

+<plusarg>
Arguments preceded with “+” are accessible by the Verilog PLI routine mc_scan_plusargs(). Optional.

+pulse_e/<percent>
Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the path delay. Optional.

A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see +pulse_r/<percent>)
propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider a path delay of 10 along with a +pulse_e/80 option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered. Note that you can force specify path delays to operate in transport mode by using the +pulse_e/0 option.

+pulse_e_style_ondetect
Selects the "on detect" style of propagating pulse errors (see +pulse_e). Optional. A pulse error propagates to the output as an X, and the "on detect" style is to schedule the X immediately, as soon as it has been detected that a pulse error has occurred. "on event" style is the default for propagating pulse errors (see +pulse_e_style_onevent).

+pulse_e_style_onevent
Selects the "on event" style of propagating pulse errors (see +pulse_e). Default. A pulse error propagates to the output as an X, and the "on event" style is to schedule the X to occur at the same time and for the same duration that the pulse would have occurred if it had propagated through normally.

+pulse_int_e/<percent>
Analogous to +pulse_e, except it applies to interconnect delays only. Optional. Used in conjunction with +multisource_int_delays (see above).

+pulse_int_r/<percent>
Analogous to +pulse_r, except it applies to interconnect delays only. Optional. Used in conjunction with +multisource_int_delays (see above).

+pulse_r/<percent>
Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the path delay. Optional.

A pulse less than the rejection limit is suppressed from propagating to the output. If the error limit is not specified by +pulse_e then it defaults to the rejection limit.

+sdf_nocheck_celltype
Disables error check for mismatch between the CELLTYPE name in the SDF file and the module or primitive name for the CELL instance. It is an error if the names do not match. Optional.

+show_canceled_e
Drives a pulse error state ('X') for the duration of a negative pulse on a specify path delay. Optional. By default ModelSim filters negative pulses.

+transport_int_delays
Selects transport mode with pulse control for single-source nets (one interconnect path). Optional. By default interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through interconnect delays. This option works independently from +multisource_int_delays.

+transport_path_delays
Selects transport mode for path delays. Optional. By default, path delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through path delays. Note that this option affects path delays only, and not primitives. Primitives always operate in inertial delay mode.
+typdelays
   Selects the typical value in min:typ:max expressions. Default. Has no effect if you
   specified the min:typ:max selection at compile time.

-v2k_int_delays
   Causes interconnect delay to be visible at the load module port. Optional. If you have
   $sdf_annotate() calls in your design that are not getting executed, add the Verilog task
   $sdf_done() after your last $sdf_annotate() to remove any zero-delay MIPDs that may
   have been created. May be used in tandem with +multisource_int_delays argument (see
   above).

Arguments, design-unit

The following library/design-unit arguments may be used with vsim. If no design-unit
specification is made, VSIM will open the Load a Design dialog box. Multiple design units
may be specified for Verilog modules and mixed VHDL/Verilog configurations.

<library_name>.<design_unit>
   Specifies a library and associated design unit; multiple library/design unit specifications
   can be made. Optional. If no library is specified, the work library is used. Environment
   variables can be used.
   The <design_unit> may be one of the following:
   <configuration>
   Specifies the VHDL configuration to simulate.
   <module> ...
   Specifies the name of one or more top-level Verilog modules to be simulated. Optional.
   <entity> [(<architecture>)]
   Specifies the name of the top-level VHDL entity to be simulated. Optional. The entity
   may have an architecture optionally specified; if omitted the last architecture compiled
   for the specified entity is simulated. An entity is not valid if a configuration is specified.

▶ Note: Most UNIX shells require arguments containing () to be single-quoted to prevent
special parsing by the shell. See the examples below.
Examples

vsim -gedge="low high" -gVCC=4.75 cpu
Invokes VSIM on the entity cpu and assigns values to the generic parameters edge and VCC. If working within the ModelSim GUI, you would enter the command as follows:

vsim (-gedge="low high") -gVCC=4.75 cpu

dism -view test=sim2.wlf
Instructs ModelSim to view the results of a previous simulation run stored in the WLF file sim2.wlf. The simulation is displayed as a dataset named "test". Use the -wlf option to specify the name of the WLF file to create if you plan to create many files for later viewing. For example:

vsim -wlf my_design.i01 my_asic structure
vsim -wlf my_design.i02 my_asic structure

dism -sdfmin /top/u1=sdf1
Annotates instance /top/u1 using the minimum timing from the SDF file sdf1.

Use multiple switches to annotate multiple instances:

vsim -sdfmin /top/u1=sdf1 -sdfmin /top/u2=sdf2 top

dism 'mylib.top(only)’ gatelib.cache_set
This example searches the libraries mylib for top(only) and gatelib for cache_set. If the design units are not found, the search continues to the work library. Specification of the architecture (only) is optional.
vsim<info>

The vsim<info> commands return information about the current vsim executable.

vsimAuth
   Returns the authorization level (PE/SE, VHDL/Verilog/PLUS).

vsimDate
   Returns the date the executable was built, such as "Apr 10 2000".

vsimId
   Returns the identifying string, such as "ModelSim 5.4".

vsimVersion
   Returns the version as used by the licensing tools, such as "1999.04".

vsimVersionString
   Returns the full vsim version string.

This same information can be obtained using the -version argument of the vsim command (CR-298).
**vsource**

The `vsource` command specifies an alternative file to use for the current source file. This command is used when the current source file has been moved. The alternative source mapping exists for the current simulation only.

**Syntax**

```
vsource
[<filename>]
```

**Arguments**

`<filename>`

Specifies a relative or full pathname. Optional. If filename is omitted the source file for the current design context is displayed.

**Examples**

```
vsource design.vhd
vsource /old/design.vhd
```
**when**

The *when* command instructs ModelSim to perform actions when the specified conditions are met. For example, you can use the *when* command to break on a signal value or at a specific simulator time (see "Time-based breakpoints" (CR-317)). Conditions can include the following HDL items: VHDL signals, and Verilog nets and registers. Use the *nowhen* command (CR-178) to deactivate *when* commands.

The *when* command uses a *when_condition_expression* to determine whether or not to perform the action. The *when_condition_expression* uses a simple restricted language (that is not related to Tcl), which permits only four operators and operands that may be either HDL item names, signame’event, or constants. ModelSim evaluates the condition every time any item in the condition changes, hence the restrictions.

With no arguments, *when* will list the currently active when statements and their labels (explicit or implicit).

▲ **Note:** Virtual signals, functions, regions, types, etc. cannot be used in the *when* command.

**Syntax**

```
when
  [ [-label <label>] [-id <id#>] ] <when_condition_expression> <command>
```

**Arguments**

- **-label <label>**
  Used to identify individual *when* commands. Optional.

- **-id <id#>**
  Attempts to assign this id number to the when command. Optional. If the id number you specify is already used, ModelSim will return an error.

▲ **Note:** Ids for when commands are assigned from the same pool as those used for the *bp* command (CR-68). So, even if you haven’t used an id number for a when command, it’s possible it is used for a breakpoint.

- **<when_condition_expression>**
  Specifies the conditions to be met for the specified *command* to be executed. Required. The condition is evaluated in the simulator kernal and can be an item name, in which case the curly braces can be omitted. The command will be executed when the item changes value. The condition can be an expression with these operators:

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>==, =</td>
</tr>
<tr>
<td>not equal</td>
<td>!=, /=</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;, AND</td>
</tr>
</tbody>
</table>
The operands may be item names, signame-event, or constants. Subexpressions in parentheses are permitted. The command will be executed when the expression is evaluated as TRUE or 1.

The formal BNF syntax is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>**</td>
</tr>
</tbody>
</table>

The operands may be item names, signame-event, or constants. Subexpressions in parentheses are permitted. The command will be executed when the expression is evaluated as TRUE or 1.

The formal BNF syntax is:

- condition ::= Name | { expression }
- expression ::= expression AND relation
  | expression OR relation
  | relation
- relation ::= Name = Literal
  | Name /= Literal
  | Name ‘ EVENT
  | { expression }
- Literal ::= '<char>' | "<bitstring>" | <bitstring>

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals, i.e., Name = Name is not possible.

The command(s) for this argument are evaluated by the Tcl interpreter within the ModelSim GUI. Any ModelSim or Tcl command or series of commands are valid with one exception—the **run** command (CR-210) cannot be used with the **when** command. Required. The command sequence usually contains a **stop** command (CR-223) that sets a flag to break the simulation run after the command sequence is completed. Multiple-line commands can be used.

**Note:** If you want to stop the simulation using a **when** command, you must use a **stop** command (CR-223) within your when statement. DO NOT use an **exit** command (CR-152) or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated. See "Ending the simulation with the stop command" (CR-316) for an example.
Examples

The **when** command below instructs the simulator to display the value of item c in binary format when there is a clock event, the clock is 1, and the value of b is 01100111, and then to stop.

```
when -label when1 {clk’event and clk='1' and b = “01100111”} {
    echo "Signal c is [exa -bin c]"
    stop
}
```

The **when** command below is labeled “a” and will cause ModelSim to echo the message “b changed” whenever the value of the item b changes.

```
when -label a b {echo "b changed"}
```

The multi-line **when** command below does not use a label and has two conditions. When the conditions are met, an **echo** (CR-143) and a **stop** (CR-223) command will be executed.

```
when {b = 1
    and c /= 0 } {
    echo "b is 1 and c is not 0"
    stop
}
```

In the example below, for the declaration "wire [15:0] a;", the **when** command will activate when the selected bits match a 7:

```
when {a(3:1) = 3'h7} {echo "matched at time" $now}
```

If you encounter a vectored net caused by compiling with **-fast**, use the ’event qualifier to prevent the command from falsely evaluating when unrelated bits of ’a’ change:

```
when {a(3:1) = 3'h7 and a(3:1)'event} {echo "matched at time" $now}
```

**Ending the simulation with the stop command**

Batch mode simulations (see "How to use checkpoint/restore" (UM-488)) are often structured as "run until condition X is true," rather than "run for X time" simulations. The multi-line **when** command below sets a done condition and executes an **echo** (CR-143) and a **stop** (CR-223) command when the condition is reached.

The simulation will not stop (even if a **quit -f** command is used) unless a **stop** command is executed. To exit the simulation and quit ModelSim, use an approach like the following:

```
onbreak {resume}
when{/done_condition == ‘1’} {
    echo "End condition reached"
    if [batch_mode] {
        set DoneConditionReached 1
        stop
    }
}
run 1000 us
if {$DoneConditionReached == 1} {
    quit -f
}
```
**Time-based breakpoints**

You can build time-based breakpoints into a `when` statement with the following syntax.

For absolute time (indicated by `@`) use:
```
when ${now = @1750ns} {stop}
```

You can also use:
```
when {errorFlag = '1' OR $now = 2ms} {stop}
```

This example adds 2ms to the simulation time at which the `when` statement is first evaluated, then stops.

You can also use variables, as shown in the following example:
```
set time 1000
when "$now = $time" {stop}
```

The quotes instruct Tcl to expand the variables before calling the command. So, the `when` command sees:
```
when "$now = 1000" stop
```

Note that "$now" has the $ escaped. This prevents Tcl from expanding the variable, because if it did, you would get:
```
when "0 = 1000" stop
```

**See also**

bp (CR-68), disablebp (CR-135), enablebp (CR-145), nowhen (CR-178)
where

The **where** command displays information about the system environment. This command is useful for debugging problems where ModelSim cannot find the required libraries or support files.

Syntax

```plaintext
where
```

Arguments

None.

Description

The **where** command displays three system settings:

- **current directory**
  
  This is the current directory that ModelSim was invoked from, or was specified on the ModelSim command line. Once in vsim, the current directory cannot be changed.

- **current project file**
  
  This is the initialization file ModelSim is using. All library mappings are taken from here. Window positions, and other parameters are taken from the `modelsim.tcl` file.
The `wlf2log` command translates a ModelSim WLF file (`vsim.wlf`) to a QuickSim II logfile. The command reads the `vsim.wlf` WLF file generated by the `add list`, `add wave`, or `log` commands in the simulator and converts it to the QuickSim II logfile format.

▲ **Important:** This command should be invoked only after you have stopped the simulation using `quit -sim` or `dataset close sim`.

### Syntax

```
wlf2log
```

### Arguments

- **-fullname**  
  Shows the full hierarchical pathname when displaying signal names. Optional.

- **-help**  
  Displays a list of command options with a brief description for each. Optional.

- **-inout**  
  Lists only the inout ports. Optional. This may be combined with the -input, -output, or -internal switches.

- **-input**  
  Lists only the input ports. Optional. This may be combined with the -output, -inout, or -internal switches.

- **-internal**  
  Lists only the internal signals. Optional. This may be combined with the -input, -output, or -inout switches.

- **-l <instance_path>**  
  Lists the signals at or below the specified HDL instance path within the design hierarchy. Optional.

- **-lower**  
  Shows all logged signals in the hierarchy. Optional. When invoked without the -lower switch, only the top level signals are displayed.

- **-o <outfile>**  
  Directs the output to be written to the file specified by `<outfile>`. Optional. The default destination for the logfile is standard out.

- **-output**  
  Lists only the output ports. Optional. This may be combined with the -input, -inout, or -internal switches.

- **-quiet**  
  Disables error message reporting. Optional.
<wlffile>
Specifies the ModelSim WLF file that you are converting. Required.

Additional information for QuickSim II users

In some cases your original QuickHDL/ModelSim simulation results (in your vsim.wlf file) may contain signal values that do not directly correspond to qsim_12state values. The resulting QuickSim II logfile generated by wlf2log may contain state values that are surrounded by single quotes, e.g. '0' and '1'. To make this logfile compatible with QuickSim models (that expect qsim_12state) you need to use a QuickSim II function named $convert_wdb().

This function was created to convert logfiles resulting from VHDL simulation that used std_logic and std_ulogic since these data types do not correlate to QuickSim's 12 simulation states. Other VHDL data types such as qsim_state or bit (2 state) do not require conversion as they are directly compatible with qsim_12state QuickSim II Waveform Databases (WDB).

The following procedure can be used to convert a wlf2log-generated logfile into a compatible QuickSim WDB. The procedure below shows how to convert the logfile while loaded into memory in QuickSim II.

1 Load the logfile (the output from wlf2log) into a WDB other than "forces". "Forces" is the default WDB, so you need to choose a unique name for the WDB when loading the logfile (for example, "fred").

2 Enter the following at the command prompt from within QuickSim:

   $convert_wdb("fred", 0)

   The first argument, which is "fred", is the name of the new WDB to be created. The second argument, which is 0, specifies the type of conversion. At this time only one type of conversion is supported. The value 0 specifies to convert std_logic or std_ulogic into qsim_12state.

3 Do a connect_wdb (either through the pulldown menus, the "Connect WDB" palette icon under "Stimulus", or a function invocation). You specify the name of the WDB that you originally loaded the logfile into (in this case, "fred").

At this point you can issue the "run" command and the stimulus in the converted logfile will be applied. Before exiting the simulation you should save the new WDB ("fred") as a WDB or logfile so that it can be loaded again in the future. The new WDB or logfile will contain the correct qsim_12state values eliminating the need to re-use convert_wdb().
wlfrecover

The **wlfrecover** tool attempts to “repair” WLF files that are incomplete due to a crash or the file being copied prior to completion of the simulation. The tool works only on WLF files created by ModelSim versions 5.6 or later. You can run the tool from the VSIM prompt or from a shell.

**Syntax**

```
wlfrecover
   <filename> [-force] [-q]
```

**Arguments**

- `<filename>`
  Specifies the WLF file to repair. Required.

- `-force`
  Disregards file locking and attempts to repair the file. Required for PCs.

- `-q`
  Hides all messages unless there is an error while repairing the file. Optional.
write cell_report

The **write cell_report** command writes to the Main window transcript or to a file a list of optimized (**-fast**) cell instances in the current design.

**Syntax**

```
write cell_report
[-filter <number>] [-infile <filename>] [-nonopt]
[-outfile] <filename>
```

**Arguments**

- **-filter <number>**
  Excludes cells with instance counts fewer than `<number>`. Optional.

- **-infile <filename>**
  Specifies a previously generated write report file to use as input. Optional. If not specified then the write report command will be run.

- **-nonopt**
  Reports only non-optimized instances. Optional.

- **[-outfile] <filename>**
  Writes the report to the specified output file rather than the Transcript. Optional.
write format

The **write format** command records the names and display options of the HDL items currently being displayed in the List or Wave window. The file created is primarily a list of **add list** (CR-48), **add wave** (CR-57), and **configure** (CR-110) commands, though a few other commands are included (see "Output" below). This file may be invoked with the **do** command (CR-138) to recreate the List or Wave window format on a subsequent simulation run.

When you load a wave or list format file, ModelSim verifies the existence of the datasets required by the format file. ModelSim displays an error message if the requisite datasets do not all exist. To force the execution of the wave or list format file even if all datasets are not present, use the **-force** switch with your **do** command. For example:

```
VSIM> do wave.do -force
```

Note that this will result in error messages for signals referencing nonexistent datasets. Also, -force is recognized by the format file not the **do** command.

**Syntax**

```
write format
  list | wave [ -window <window_name>] <filename>
```

**Arguments**

- **list | wave**
  
  Specifies that the contents of either the List or the Wave window are to be recorded. Required.

- **-window <window_name>**
  
  Specifies the window for which you want contents recorded. Optional. Use when you have more than one instance of the List or Wave window.

- **<filename>**
  
  Specifies the name of the output file where the data is to be written. Required.

**Examples**

```
write format list alu_list.do
  Saves the current data in the List window in a file named alu_list.do.

write format wave alu_wave.do
  Saves the current data in the Wave window in a file named alu_wave.do.
```

**Output**

Below is an example of a saved Wave window format file.

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic /cntr_struct/ld
add wave -noupdate -format Logic /cntr_struct/rst
add wave -noupdate -format Logic /cntr_struct/clk
add wave -noupdate -format Literal /cntr_struct/d
add wave -noupdate -format Literal /cntr_struct/q
```
TreeUpdate [SetDefaultTree]
quietly WaveActivateNextPane
add wave -noupdate -format Logic /cntr_struct/p1
add wave -noupdate -format Logic /cntr_struct/p2
add wave -noupdate -format Logic /cntr_struct/p3
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {0 ns}
WaveRestoreZoom {0 ns} {1 us}
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -signalnamewidth 0
configure wave -justifyvalue left

In the example above, five signals are added with the -noupdate argument to the default window pane. The TreeUpdate command then refreshes all five waveforms. The second WaveActivateNextPane command creates a second pane which contains three signals. The WaveRestoreCursors command restores any cursors you set during the original simulation, and the WaveRestoreZoom command restores the Zoom range you set. These four commands are used only in saved Wave format files; therefore, they are not documented elsewhere.

See also

add list (CR-48), add wave (CR-57)
write list

The **write list** command records the contents of the most recently opened or specified List window in a list output file. This file contains simulation data for all HDL items displayed in the List window: VHDL signals and variables and Verilog nets and registers.

**Syntax**

```
write list
   [-events] [-window <wname>] <filename>
```

**Arguments**

- **-events**
  Specifies to write print-on-change format. Optional. Default is tabular format.

- **-window <wname>**
  Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-263) to change the default window.

- **<filename>**
  Specifies the name of the output file where the data is to be written. Required.

**Examples**

```
write list alu.lst
   Saves the current data in the default List window in a file named *alu.lst*.

write list -win list1 group1.list
   Saves the current data in window ‘list1’ in a file named *group1.list*.
```

**See also**

`write tssi` (CR-329)
write preferences

The **write preferences** command saves the current GUI preference settings to a Tcl preference file. Settings saved include current window locations and sizes; Wave, Signals and Variables window column widths; Wave, Signals and Variables window value justification; and Wave window signal name width.

**Syntax**

```
write preferences
   <preference file name>
```

**Arguments**

`<preference file name>`

Specifies the name for the preference file. Optional. If the file is named `modelsim.tcl`, ModelSim will read the file each time vsim is invoked. To use a preference file other than `modelsim.tcl` you must specify the alternative file name with the `MODELSIM_TCL` (UM-442) environment variable.

**See also**

You can modify variables by editing the preference file with the ModelSim `notepad` (CR-176):

```
notepad <preference file name>
```
write report

The `write report` command prints a summary of the design being simulated including a list of all design units (VHDL configurations, entities, and packages and Verilog modules) with the names of their source files. If you have compiled a Verilog design using `-fast` (see "Compiling for faster performance" (UM-99)), the report will also identify cells which have been optimized.

Syntax

```plaintext
write report
    [[<filename>] [-l | -s]] | [-tcl]
```

Arguments

- `<filename>`
  Specifies the name of the output file where the data is to be written. Optional. If the `<filename>` is omitted, the report is written to the Main window.

- `-l`
  Generates more detailed information about the design. Default.

- `-s`
  Generates a short list of design information. Optional

- `-tcl`
  Generates a Tcl list of design unit information. Optional. This argument cannot be used with a filename.

Examples

```plaintext
write report alu.rep
    Saves information about the current design in a file named `alu.rep`.```
write transcript

The **write transcript** command writes the contents of the Main window Transcript to the specified file. The resulting file can be used to replay the transcribed commands as a DO file (macro).

**Syntax**

```
write transcript
    [<filename>]
```

**Arguments**

- `<filename>`
  Specifies the name of the output file where the data is to be written. Optional. If the `<filename>` is omitted, the transcript is written to a file named `transcript`.

**See also**

- **do** (CR-138)
**write tssi**

The `write tssi` command records the contents of the default or specified List window in a "TSSI format" file. The file contains simulation data for all HDL items displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). A signal definition file is also generated.

The List window needs to be using symbolic radix in order for `write tssi` to produce useful output.

**Syntax**

```
write tssi
   [-window <wname>] <filename>
```

**Arguments**

- `-window <wname>`
  Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the `view` command (CR-263) to change the default window.

- `<filename>`
  Specifies the name of the output file where the data is to be written. Required.

**Description**


If the `<filename>` has a file extension (e.g., `listfile.lst`), then the definition file is given the same file name with the extension .def (e.g., `listfile.def`). The values in the listfile are produced in the same order that they appear in the List window. The directionality is determined from the port type if the item is a port, otherwise it is assumed to be bidirectional (mode INOUT).

Items that can be converted to SEF are VHDL enumerations with 255 or fewer elements and Verilog nets. The enumeration values U, X, 0, 1, Z, W, L, H and - (the enumeration values defined in the IEEE Standard 1164 `std_ulogic` enumeration) are converted to SEF values according to the table below. Other values are converted to a question mark (?) and cause an error message. Though the write tssi command was developed for use with `std_ulogic`, any signal which uses only the values defined for `std_ulogic` (including the VHDL standard type `bit`) will be converted.

<table>
<thead>
<tr>
<th>std_ulogic State Characters</th>
<th>SEF State Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input</td>
</tr>
<tr>
<td>U</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

Bidirectional logic values are not converted because only the resolved value is available. The Fluence (TSSI) TDS ASCII In Converter and ASCII Out Converter can be used to resolve the directionality of the signal and to determine the proper forcing or expected value on the port. Lowercase values x, z, w, l and h are converted to the same values as the corresponding capitalized values. Any other values will cause an error message to be generated the first time an invalid value is detected on a signal, and the value will be converted to a question mark (?).

▶ **Note:** The TDS ASCII In Converter and ASCII Out Converter are part of the TDS software from Fluence Technology. ModelSim outputs a vector file, and Fluence’s tools determine whether the bidirectional signals are driving or not.

### See also

`tssi2mti` (CR-230)
**write wave**

The `write wave` command records the contents of the most currently opened or specified Wave window in PostScript format. The output file can then be printed on a PostScript printer.

**Syntax**

```
write wave
[ -window <wname> ] [ -width <real_num> ] [ -height <real_num> ]
[ -margin <real_num> ] [ -start <time> ] [ -end <time> ] [ -perpage <time> ]
[ -pagecount <n> ] [ -landscape ] [ -portrait ] <filename>
```

**Arguments**

- `-window <wname>`
  Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the `view` command (CR-263) to change the default window.

- `-width <real_num>`
  Specifies the paper width in inches. Optional. Default is 8.5.

- `-height <real_num>`
  Specifies the paper height in inches. Optional. Default is 11.0.

- `-margin <real_num>`
  Specifies the margin in inches. Optional. Default is 0.5.

- `-start <time>`
  Specifies the start time (on the waveform time scale) to be written. Optional.

- `-end <time>`
  Specifies the end time (on the waveform time scale) to be written. Optional.

- `-perpage <time>`
  Specifies the time width per page of output. Optional.

- `-pagecount <n>`
  Specifies the number of pages to use horizontally along the time axis. Optional.

- `-landscape`
  Use landscape (horizontal) orientation. Optional. This is the default orientation.

- `-portrait`
  Use portrait (vertical) orientation. Optional. The default is landscape (horizontal).

- `<filename>`
  Specifies the name of the PostScript output file. Required.

**Examples**

```
write wave alu.ps
  Saves the current data in the Wave window in a file named `alu.ps`.
write wave -win wave2 group2.ps
  Saves the current data in window ‘wave2’ in a file named `group2.ps`.
```
write wave -start 600ns -end 800ns -perpage 100ns top.ps

Writes two separate pages to top.ps as indicated in the illustration (the actual PostScript print out will show all items listed in the Wave window, not just the portion in view):

– page 1 starts at 600 ns
– page 2 ends at 800 ns

To make the job of creating a PostScript waveform output file easier, use the File > Print Postscript menu selection in the Wave window. See "Printing and saving waveforms" (UM-276) for more information.
Licensing Agreement

IMPORTANT – USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS

CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE

This license is a legal “Agreement” concerning the use of Software between you, the end user, either individually or as an authorized representative of the company purchasing the license, and Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Mentor Graphics (Singapore) Private Limited, and their majority-owned subsidiaries (“Mentor Graphics”). USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. If you do not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within 10 days after receipt of Software and receive a full refund of any license fee paid.

END USER LICENSE AGREEMENT

1. GRANT OF LICENSE. The software programs you are installing, downloading, or have acquired with this Agreement, including any updates, modifications, revisions, copies, and documentation (“Software”) are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics or its authorized distributor grants to you, subject to payment of appropriate license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form; (b) for your internal business purposes; and (c) on the computer hardware or at the site for which an applicable license fee is paid, or as authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Mentor Graphics’ then-current standard policies, which vary depending on Software, license fees paid or service plan purchased, apply to the following and are subject to change: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be communicated and technically implemented through the use of authorization codes or similar devices); (c) eligibility to receive updates, modifications, and revisions; and (d) support services provided. Current standard policies are available upon request.

2. ESD SOFTWARE. If you purchased a license to use embedded software development (“ESD”) Software, Mentor Graphics or its authorized distributor grants to you a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESD compilers, including the ESD run-time libraries distributed with ESD C and C++ compiler Software that are linked into a composite program as an integral part of your compiled computer program, provided that you distribute these files only in conjunction with your compiled computer program. Mentor Graphics does NOT grant you any right to duplicate or incorporate copies of Mentor Graphics’ real-time operating systems or other ESD Software, except those explicitly granted in this section, into your products without first signing a separate agreement with Mentor Graphics for such purpose.
3. BETA CODE.

3.1 Portions or all of certain Software may contain code for experimental testing and evaluation (“Beta Code”), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics’ authorization, Mentor Graphics grants to you a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and your use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

3.2 If Mentor Graphics authorizes you to use the Beta Code, you agree to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. You will contact Mentor Graphics periodically during your use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of your evaluation and testing, you will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

3.3 You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceives or makes during or subsequent to this Agreement, including those based partly or wholly on your feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this subsection shall survive termination or expiration of this Agreement.

4. RESTRICTIONS ON USE. You may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. You shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. You shall not make Software available in any form to any person other than your employer's employees and contractors, excluding Mentor Graphics' competitors, whose job performance requires access. You shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access to Software does not disclose it or use it except as permitted by this Agreement. Except as otherwise permitted for purposes of interoperability as specified by the European Union Software Directive or local law, you shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code. You may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it without Mentor Graphics’ prior written consent. The provisions of this section shall survive the termination or expiration of this Agreement.

5. LIMITED WARRANTY.

5.1 Mentor Graphics warrants that during the warranty period Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will
meet your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. You must notify Mentor Graphics in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or installation.

MENTOR GRAPHICS’ ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS’ OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LOANED TO YOU FOR A LIMITED TERM OR AT NO COST; OR (C) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED “AS IS.”

5.2 THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6. LIMITATION OF LIABILITY. EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE STATUTE OR REGULATION, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS’ OR ITS LICENSORS’ LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER.

7. LIFE ENDANGERING ACTIVITIES. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. YOU AGREE TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS’ FEES, ARISING OUT OF OR IN CONNECTION WITH SUCH USE.

8. INFRINGEMENT.

8.1 Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright in the United States, Canada, Japan, Switzerland, Norway, Israel, Egypt, or the
European Union. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the claim, provided that you: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the claim; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the claim.

8.2 If an infringement claim is made, Mentor Graphics may, at its option and expense, either (a) replace or modify Software so that it becomes noninfringing, or (b) procure for you the right to continue using Software. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

8.3 Mentor Graphics has no liability to you if the alleged infringement is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that you design or market; (f) any Beta Code contained in Software; or (g) any Software provided by Mentor Graphics’ licensors which do not provide such indemnification to Mentor Graphics’ customers.

8.4 THIS SECTION 8 STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND YOUR SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

9. TERM. This Agreement remains effective until expiration or termination. This Agreement will automatically terminate if you fail to comply with any term or condition of this Agreement or if you fail to pay for the license when due and such failure to pay continues for a period of 30 days after written notice from Mentor Graphics. If Software was provided for limited term use, this Agreement will automatically expire at the end of the authorized term. Upon any termination or expiration, you agree to cease all use of Software and return it to Mentor Graphics or certify deletion and destruction of Software, including all copies, to Mentor Graphics’ reasonable satisfaction.

10. EXPORT. Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. You agree that you will not export in any manner any Software or direct product of Software, without first obtaining all necessary approval from appropriate local and United States government agencies.

11. RESTRICTED RIGHTS NOTICE. Software has been developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial
12. THIRD PARTY BENEFICIARY. For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth in this Agreement.

13. CONTROLLING LAW. This Agreement shall be governed by and construed under the laws of Ireland if the Software is licensed for use in Israel, Egypt, Switzerland, Norway, South Africa, or the European Union, the laws of Japan if the Software is licensed for use in Japan, the laws of Singapore if the Software is licensed for use in Singapore, People’s Republic of China, Republic of China, India, or Korea, and the laws of the state of Oregon if the Software is licensed for use in the United States of America, Canada, Mexico, South America or anywhere else worldwide not provided for in this section.

14. SEVERABILITY. If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

15. MISCELLANEOUS. This Agreement contains the entire understanding between the parties relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions, except valid license agreements related to the subject matter of this Agreement which are physically signed by you and an authorized agent of Mentor Graphics. This Agreement may only be modified by a physically signed writing between you and an authorized agent of Mentor Graphics. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse. The prevailing party in any legal action regarding the subject matter of this Agreement shall be entitled to recover, in addition to other relief, reasonable attorneys' fees and expenses.

Rev. 03/00
Index


Symbols

+acc option, design object visibility UM-105
+opt UM-100
+typdelays CR-294
.so, shared object file
  loading PLI/VPI C applications UM-124
  loading PLI/VPI C++ applications UM-127

Numerics

1076, IEEE Std UM-20
1364, IEEE Std UM-20, UM-81
64-bit ModelSim, using with 32-bit FLI apps UM-129

A

+acc option, design object visibility UM-105
deltas
  explained UM-513
abort command CR-44
absolute time, using @ CR-17
ACC routines UM-137
accelerated packages UM-55
access
  hierarchical items UM-357
  limitations in mixed designs UM-144
add button command CR-45
add list command CR-48
add wave command CR-57
add_menu command CR-51
add_menucb command CR-53
add_menuitem simulator command CR-54
add_separator command CR-55
add_submenu command CR-56
alias command CR-61
application notes UM-24
architecture simulator state variable UM-456
argc simulator state variable UM-456
arrays
  indexes CR-15
  slices CR-15
AssertFile .ini file variable UM-446
AssertionFormat .ini file variable UM-447
assertions
  messages, turning off UM-452
  selecting severity that stops simulation UM-298
testing for using a DO file UM-494
attributes, of signals, using in expressions CR-24

B

bad magic number error message UM-155
balloon dialog, toggling on/off UM-266
base (radix), specifying in List window UM-209
batch_mode command CR-62
batch-mode simulations UM-490, UM-491
  halting CR-316
bd (breakpoint delete) command CR-63
binary radix, mapping to std_logic values CR-21
blocking assignments UM-94
bookmark add wave command CR-64
bookmark delete wave command CR-65
bookmark goto wave command CR-66
bookmark list wave command CR-67
bookmarks UM-272
bp (breakpoint) command CR-68
break
  on assertion UM-298
  on signal value CR-314
  stop simulation run UM-182, UM-234
BreakOnAssertion .ini file variable UM-447, UM-454
breakpoints
  conditional CR-314, UM-228
  continuing simulation after CR-210
  deleting CR-63, UM-235, UM-301
  listing CR-68
  setting CR-68, UM-235
  signal breakpoints (when statements) CR-314, UM-228
  Source window, viewing in UM-229
  time-based UM-228
    in when statements CR-317
.bs file UM-202
buffered/unbuffered output UM-449
bus contention checking UM-498
  configuring CR-75
  disabling CR-76
  enabling CR-74
bus float checking UM-498
  configuring CR-78
  disabling CR-79
  enabling CR-77
busses
RTL-level, reconstructing UM-162
user-defined CR-58, UM-210, UM-259
Button Adder (add buttons to windows) UM-310
buttons, adding to the Main window toolbar CR-45

C

C applications
  compiling and linking UM-124
C++ applications
  compiling and linking UM-127
case choice, must be locally static CR-254
case sensitivity
  named port associations UM-152
  VHDL vs. Verilog CR-15
causality, tracing in Dataflow window UM-196
cd (change directory) command CR-71
cell libraries UM-111
change command CR-72
change_menu_cmd command CR-73
chasing X UM-197
check contention add command CR-74
check contention config command CR-75
check contention off command CR-76
check float add command CR-77
check float config command CR-78
check float off command CR-79
check stable off command CR-80
check stable on command CR-81
checkpoint command CR-82
checkpoint/restore UM-488
CheckpointCompressMode .ini file variable UM-447, UM-454
CheckSynthesis .ini file variable UM-445
clear differences UM-352
clock change, sampling signals at UM-494
clocked comparison UM-341, UM-347
Code Coverage
  coverage clear command CR-116
  coverage data in Source window UM-334
  coverage exclude clear command CR-117
  coverage exclude disable command CR-118
  coverage exclude enable command CR-119
  coverage exclude load command CR-120
  coverage reload command CR-121
  coverage report command CR-122
  coverage_summary window UM-330
  enabling code coverage UM-330, UM-338
  excluding lines/files UM-332, UM-335
  merging report files CR-121, UM-336
  miss and exclusion details UM-331
  saving coverage reports UM-333
  Tcl preference variables UM-338
  combining signals, user-defined bus CR-58, UM-210, UM-259
  command history UM-178
  command reference UM-23
  CommandHistory .ini file variable UM-447
  command-line mode UM-490
  commands
    .main clear CR-37
    .wave.tree interrupt CR-38
    .wave.tree zoomfull CR-39
    .wave.tree zoomin CR-40
    .wave.tree zoomlast CR-41
    .wave.tree zoomout CR-42
    .wave.tree zoomrange CR-43
    abort CR-44
    add button CR-45
    add list CR-48
    add wave CR-57
    add_menu CR-51
    add_menucb CR-53
    add_menuitem CR-54
    add_separator CR-55
    add_submenu CR-56
    alias CR-61
    batch_mode CR-62
    bd (breakpoint delete) CR-63
    bookmark add wave CR-64
    bookmark delete wave CR-65
    bookmark goto wave CR-66
    bookmark list wave CR-67
    bp (breakpoint) CR-68
    cd (change directory) CR-71
    change CR-72
    change_menu_cmd CR-73
    check contention add CR-74
    check contention config CR-75
    check contention off CR-76
    check float add CR-77
    check float config CR-78
    check float off CR-79
    check stable off CR-80
    check stable on CR-81
    checkpoint CR-82
...
compare info CR-93
compare list CR-95
compare open CR-106
compare options CR-96
compare region CR-83
compare reload CR-99
compare savediffs CR-102
compare saverrules CR-103
compare see CR-104
compare start CR-101
configure CR-110
coverage clear CR-116
coverage exclude clear CR-117
coverage exclude disable CR-118
coverage exclude enable CR-119
coverage exclude load CR-120
coverage reload CR-121
coverage report CR-122
dataset alias CR-123
dataset clear CR-124
dataset close CR-125
dataset info CR-126
dataset list CR-127
dataset open CR-128
dataset rename CR-129, CR-130
dataset snapshot CR-131
delete CR-133
describe CR-134
disable_menu CR-136
disable_menuitem CR-137
disablebp CR-135
do CR-138
down CR-139
drivers CR-141
dumplog64 CR-142
echo CR-143
edit CR-144
enable_menu CR-146
enable_menuitem CR-147
enablebp CR-145
environment CR-148
examine CR-149
exit CR-152
find CR-153
force CR-156
getactivecursortime CR-159
getactivemarkertime CR-160
graphic interface commands UM-317
help CR-161
history CR-162
lecho CR-163
left CR-164
log CR-166
lshift CR-168
lsublist CR-169
macro_option CR-170
modelsim CR-171
next CR-172
noforce CR-173
nolog CR-174
notation conventions CR-10
notepad CR-176
noview CR-177
nowhen CR-178
onbreak CR-179
onElabError CR-180
onerror CR-181
pause CR-182
play CR-183
power add CR-184
power report CR-185
power reset CR-186
printenv CR-187
profile clear CR-188
profile interval CR-189
profile off CR-190
profile on CR-191
profile option CR-192
profile report CR-193
property list CR-195
property wave CR-196
pwd CR-197
quietly CR-198
quit CR-199
radix CR-200
record CR-201
report CR-202
restart CR-204
restore CR-206
resume CR-207
right CR-208
run CR-210
search CR-212
searchlog CR-214
seetime CR-216
shift CR-217
show CR-218
splitio CR-220
status CR-221
step CR-222
stop CR-223
system UM-427
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tb (traceback)</td>
<td>CR-224</td>
</tr>
<tr>
<td>toggle add</td>
<td>CR-225</td>
</tr>
<tr>
<td>toggle report</td>
<td>CR-226</td>
</tr>
<tr>
<td>toggle reset</td>
<td>CR-227</td>
</tr>
<tr>
<td>transcribe</td>
<td>CR-228</td>
</tr>
<tr>
<td>transcript</td>
<td>CR-229</td>
</tr>
<tr>
<td>TreeUpdate</td>
<td>CR-324</td>
</tr>
<tr>
<td>tssi2mti</td>
<td>CR-230</td>
</tr>
<tr>
<td>up</td>
<td>CR-231</td>
</tr>
<tr>
<td>variables referenced in</td>
<td>CR-17</td>
</tr>
<tr>
<td>vcd add</td>
<td>CR-233</td>
</tr>
<tr>
<td>vcd checkpoint</td>
<td>CR-234</td>
</tr>
<tr>
<td>vcd comment</td>
<td>CR-235</td>
</tr>
<tr>
<td>vcd dumpports</td>
<td>CR-236</td>
</tr>
<tr>
<td>vcd dumpportsall</td>
<td>CR-238</td>
</tr>
<tr>
<td>vcd dumpportsflush</td>
<td>CR-239</td>
</tr>
<tr>
<td>vcd dumpportslimit</td>
<td>CR-240</td>
</tr>
<tr>
<td>vcd dumpportsoff</td>
<td>CR-241</td>
</tr>
<tr>
<td>vcd dumpportson</td>
<td>CR-242</td>
</tr>
<tr>
<td>vcd file</td>
<td>CR-243</td>
</tr>
<tr>
<td>vcd files</td>
<td>CR-245</td>
</tr>
<tr>
<td>vcd flush</td>
<td>CR-247</td>
</tr>
<tr>
<td>vcd limit</td>
<td>CR-248</td>
</tr>
<tr>
<td>vcd off</td>
<td>CR-249</td>
</tr>
<tr>
<td>vcd on</td>
<td>CR-250</td>
</tr>
<tr>
<td>vcom</td>
<td>CR-252</td>
</tr>
<tr>
<td>vdel</td>
<td>CR-258</td>
</tr>
<tr>
<td>vdir</td>
<td>CR-259</td>
</tr>
<tr>
<td>verror</td>
<td>CR-260</td>
</tr>
<tr>
<td>vgencomp</td>
<td>CR-261</td>
</tr>
<tr>
<td>view</td>
<td>CR-263</td>
</tr>
<tr>
<td>virtual count</td>
<td>CR-265</td>
</tr>
<tr>
<td>virtual define</td>
<td>CR-266</td>
</tr>
<tr>
<td>virtual delete</td>
<td>CR-267</td>
</tr>
<tr>
<td>virtual describe</td>
<td>CR-268</td>
</tr>
<tr>
<td>virtual expand</td>
<td>CR-269</td>
</tr>
<tr>
<td>virtual function</td>
<td>CR-270</td>
</tr>
<tr>
<td>virtual hide</td>
<td>CR-273</td>
</tr>
<tr>
<td>virtual log</td>
<td>CR-274</td>
</tr>
<tr>
<td>virtual nobide</td>
<td>CR-276</td>
</tr>
<tr>
<td>virtual nolog</td>
<td>CR-277</td>
</tr>
<tr>
<td>virtual region</td>
<td>CR-279</td>
</tr>
<tr>
<td>virtual save</td>
<td>CR-280</td>
</tr>
<tr>
<td>virtual show</td>
<td>CR-281</td>
</tr>
<tr>
<td>virtual signal</td>
<td>CR-282</td>
</tr>
<tr>
<td>virtual type</td>
<td>CR-285</td>
</tr>
<tr>
<td>vlib</td>
<td>CR-287</td>
</tr>
<tr>
<td>vlog</td>
<td>CR-288</td>
</tr>
<tr>
<td>vmake</td>
<td>CR-296</td>
</tr>
<tr>
<td>vmap</td>
<td>CR-297</td>
</tr>
<tr>
<td>vsim</td>
<td>CR-298</td>
</tr>
<tr>
<td>VSIM Tcl commands</td>
<td>UM-428</td>
</tr>
<tr>
<td>vsimDate</td>
<td>CR-312</td>
</tr>
<tr>
<td>vsimId</td>
<td>CR-312</td>
</tr>
<tr>
<td>vsimVersion</td>
<td>CR-312</td>
</tr>
<tr>
<td>WaveActivateNextPane</td>
<td>CR-324</td>
</tr>
<tr>
<td>WaveRestoreCursors</td>
<td>CR-324</td>
</tr>
<tr>
<td>WaveRestoreZoom</td>
<td>CR-324</td>
</tr>
<tr>
<td>when</td>
<td>CR-314</td>
</tr>
<tr>
<td>where</td>
<td>CR-318</td>
</tr>
<tr>
<td>wlf2log</td>
<td>CR-319</td>
</tr>
<tr>
<td>wlfrecover</td>
<td>CR-321</td>
</tr>
<tr>
<td>write cell_report</td>
<td>CR-322</td>
</tr>
<tr>
<td>write format</td>
<td>CR-323</td>
</tr>
<tr>
<td>write list</td>
<td>CR-325</td>
</tr>
<tr>
<td>write preferences</td>
<td>CR-326</td>
</tr>
<tr>
<td>write report</td>
<td>CR-327</td>
</tr>
<tr>
<td>write transcript</td>
<td>CR-328</td>
</tr>
<tr>
<td>write tssi</td>
<td>CR-329</td>
</tr>
<tr>
<td>write wave</td>
<td>CR-331</td>
</tr>
<tr>
<td>comment characters in VSIM commands</td>
<td>CR-10</td>
</tr>
</tbody>
</table>
| compare | |}

**Compare**
- add region UM-346
- add signals UM-345
- by signal UM-345
- clear differences UM-352
- clocked UM-341, UM-347
- continuous UM-341, UM-348
- difference markers UM-350
- differences UM-353
- displayed in List window UM-354
- end UM-352
- graphic interface UM-343
- icons UM-351
- limit count UM-349
- menu UM-352
- modes UM-341
- options UM-349
- pathnames UM-350
- preference variables UM-355
- reference dataset UM-343
- reference region UM-346
- reload UM-353
- rules UM-353
- run UM-352
- save differences UM-353
- show differences UM-352
- specify dataset UM-343
- start UM-352
- startup wizard UM-352
- tab UM-344
- test dataset UM-343
test region UM-346
timing differences UM-350
tolerance UM-348
tolerances UM-341
values UM-351
verilog matching UM-349
VHDL matching UM-349
wave window display UM-350
waveforms UM-339
wizard UM-352
write report UM-353

compare annotate command CR-86, CR-89
compare by region UM-346
compare clock command CR-87
compare close command CR-92
compare commands UM-355
compare delete command CR-91
compare info command CR-93
compare list command CR-95
compare open command CR-106
compare options command CR-96
compare region command CR-83
compare reload command CR-99
compare savediffs command CR-102
compare saverules command CR-103
compare see command CR-104
compare simulations UM-153
compare start command CR-101
compatibility, of vendor libraries CR-259
compile history UM-37
compile order
  auto generate UM-39
  changing UM-39
compiler directives UM-119
  IEEE Std 1364-2000 UM-119
  XL compatible compiler directives UM-120
Compiling
  with the graphic interface UM-282
compiling
  +opt argument UM-100
  changing order in the GUI UM-39
  compile history UM-37
  default options, setting UM-284
  -fast argument UM-99
  grouping files UM-40
  options, in projects UM-45
  order, changing in projects UM-39
  range checking in VHDL CR-255, UM-61
  source errors, locating UM-283
  Verilog CR-288, UM-82
    incremental compilation UM-83
    library components, including CR-290
    optimizing performance CR-289, UM-99
    XL `uselib compiler directive UM-87
    XL compatible options UM-86
VHDL CR-252, UM-61
  at a specified line number CR-253
  selected design units (-just eapbc) CR-253
  standard package (-s) CR-256
VITAL packages UM-73
  with the graphic interface UM-282
component declaration
  generating VHDL from Verilog UM-150
  vgencomp UM-150
concatenation
  directives CR-19
  of signals CR-19, CR-282
ConcurrentFileLimit.ini file variable UM-447
conditional breakpoints CR-314, UM-228
configuration simulator state variable UM-456
configurations, simulating CR-298
configure command CR-110
connectivity, exploring UM-193
constants
  examining in a package UM-497
  in case statements CR-254
  values of, displaying CR-134, CR-149
context menus
  coverage_source window UM-335
    described UM-170
    Library tab UM-51
    Project tab UM-37
    Structure pages UM-240
continuous comparison UM-341
convert real to time UM-77
convert time to real UM-76
coverage clear command CR-116
coverage exclude clear command CR-117
coverage exclude disable command CR-118
coverage exclude enable command CR-119
coverage exclude load command CR-120
coverage reload command CR-121
coverage report command CR-122
coverage_summary window UM-330
cursors
  link to Dataflow window UM-187
  trace events with UM-196
  Wave window UM-269
customizing
  adding buttons CR-45
  via preference variables UM-454
D

Dataflow window UM-186
  pan UM-195
  zoom UM-195
  see also windows, Dataflow window
dataflow.bsm file UM-202
dataset alias command CR-123
Dataset Browser UM-157
dataset clear command CR-124
dataset close command CR-125
dataset info command CR-126
dataset list command CR-127
dataset open command CR-128
dataset rename command CR-129, CR-130
Dataset Snapshot UM-159
dataset snapshot command CR-131
datasets UM-153, UM-340
  environment command, specifying with CR-148
  managing UM-157
  reference UM-343
  restrict dataset prefix display UM-158
  simulator resolution UM-154
  specifying for compare UM-343
  test UM-343
DatasetSeparator.ini file variable UM-447
Debug Detective UM-305
declarations, hiding implicit with explicit CR-257
default compile options UM-284
default editor, changing UM-441
DefaultForceKind.ini file variable UM-447, UM-454
DefaultRadix.ini file variable UM-447, UM-454
DefaultRestartOptions variable UM-447, UM-453
defaults
  restoring UM-441
  window arrangement UM-170
+define+ CR-289
delay
  delta delays UM-513
  infinite zero-delay loops, detecting UM-500
  interconnect CR-302
  modes for Verilog models UM-111
  SDF files UM-377
  stimulus delay, specifying UM-226
+delay_mode_distributed CR-289
+delay_mode_path CR-289
+delay_mode_unit CR-289
+delay_mode_zero CR-289
'delayed CR-24
DelayFileOpen.ini file variable UM-448, UM-455
delete command CR-133
delimiters UM-50
delta simulator state variable UM-456
deltas
  collapsing in the List window UM-212
  hiding in the List window CR-111, UM-212
  infinite zero-delay loops UM-500
  referencing simulator iteration
    as a simulator state variable UM-456
dependent design units UM-61
describe command CR-134
descriptions of HDL items UM-235
design hierarchy, viewing in Structure window UM-237
design library
  creating UM-49
  logical name, assigning UM-52
  mapping search rules UM-53
  resource type UM-48
  VHDL design units UM-61
  working type UM-48
design stability checking UM-499
design units UM-48
  hierarchy of, viewing UM-171
  report of units simulated CR-327
  Verilog
    adding to a library CR-288
directories
  mapping libraries CR-297
  moving libraries UM-53
disable_menu command CR-136
disable_menuitem command CR-137
disablebp command CR-135
distributed delay mode UM-112
dividers, in Wave window UM-257
DLL files, loading UM-124, UM-127
do command CR-138
DO files (macros) CR-138
  error handling UM-437
  executing at startup UM-441, UM-449
  parameters, passing to UM-435
  Tcl source command UM-438
documentation UM-24
DOPATH environment variable UM-441
down command CR-139
Drivers
  Dataflow Window UM-193
  drivers
    show in Dataflow window UM-260
    Wave window UM-260
  drivers command CR-141
  drivers, multiple on unresolved signal UM-285
dump files, viewing in ModelSim CR-251
dumplog64 command CR-142
dumpports tasks, VCD files UM-392

E

echo command CR-143
edges, finding CR-164, CR-208
edit command CR-144
Editing
in notepad windows UM-183, UM-462
in the Main window UM-183, UM-462
in the Source window UM-183, UM-462
EDITOR environment variable UM-441
editor, default, changing UM-441
elab_defer_fli argument UM-65, UM-110
elaboration file
creating UM-63, UM-108
loading UM-64, UM-109
modifying stimulus UM-64, UM-109
resimulating the same design UM-63, UM-108
simulating with PLI or FLI models UM-65, UM-110
elaboration, interrupting CR-298
embedded wave viewer UM-194
enable_menu command CR-146
enable_menuitem command CR-147
enablebp command CR-145
encryption, securing pre-compiled libraries UM-492
end comparison UM-352
ENDFILE function UM-69
ENDLINE function UM-69
entities, specifying for simulation CR-310
entity simulator state variable UM-456
enumerated types UM-495
user defined CR-285
environment command CR-148
environment variables UM-441
accessed during startup UM-483
license file UM-474
reading into Verilog code CR-289
referencing from ModelSim command line UM-443
referencing with VHDL FILE variable UM-443
setting in Windows UM-442
specifying library locations in modelsim.ini file UM-444
specifying UNIX editor CR-144
transcript file, specifying location of UM-449
used in Solaris linking for FLI UM-125
using in pathnames CR-14
using with location mapping UM-501

variable substitution using Tcl UM-427
viewing current names and values with printenv CR-187
environment, displaying or changing pathname CR-148
error messages
bad magic number UM-155
getting more information UM-466
Tcl_init error UM-470
events
during compilation, locating UM-283
getting details about messages CR-260
onerror command CR-181
event order
changing in Verilog CR-288
in optimized designs UM-107
in Verilog simulation UM-92
event queues UM-92
events, tracing UM-196
examine command CR-149
examine tooltip
toggling on/off UM-266
examining constants in a package UM-497
exclusion filter UM-332
ext codes UM-468
exit command CR-152
expand net UM-193
Explicit .ini file variable UM-445
Expression Builder UM-305, UM-347
specify when expression UM-347, UM-348
Expression_format CR-18
extended identifiers UM-149
syntax in commands CR-15

F

-f CR-289
F8 function key UM-185, UM-464
-fast CR-289, UM-99
feature names, described UM-476
features, new UM-545
file-line breakpoints UM-235
files, grouping for compile UM-40
Find
cursors in Wave window UM-169
specified time in Wave window UM-169
time markers in List window UM-169
find command CR-153
finding
cursor in the Wave window UM-270
marker in the List window UM-216
names and values UM-169
FLEXlm license manager UM-473–UM-479
administration tools for Windows UM-479
license server utilities UM-478
FLI UM-78
folders, in projects UM-43
force command CR-156
defaults UM-453
foreign language interface UM-78
format file
  List window CR-323
  Wave window CR-323, UM-248
FPGA libraries, importing UM-57
frequently asked questions UM-24

G
gate-level designs, optimizing UM-101
GenerateFormat .ini file variable UM-448
generics
  assigning or overriding values with -g and -G CR-300
  examining generic values CR-149
  limitation on assigning composite types CR-300
  VHDL UM-145
get_resolution() VHDL function UM-74
getactivetimes command CR-159
getactivemarkertime command CR-160
graphic interface UM-165–??
  UNIX support UM-19
  grouping files for compile UM-40
GUI_expression_format CR-18
GUI expression builder UM-305
syntax CR-22

H
halting waveform drawing CR-38
hardware model interface UM-414
'hasX CR-24
Hazard .ini file variable (VLOG) UM-446
hazards
  -hazards argument to vlog CR-290
  -hazards argument to vsim CR-307
  limitations on detection UM-95
HDL item UM-23
help command CR-161
hierarchical profile, Performance Analyzer UM-322
hierarchical references, mixed-language UM-144
hierarchy
driving signals in UM-359, UM-368
forcing signals in UM-75, UM-364, UM-373
referencing signals in UM-75, UM-362, UM-371
releasing signals in UM-75, UM-366, UM-375
viewing signal names without UM-265
history
  of commands
    shortcuts for reuse CR-11, UM-461
  of compiles UM-37
history command CR-162
hm_entity UM-415
HOME environment variable UM-441
Hotkey
  g - display specified time in Wave window UM-169
Hotkeys UM-274, UM-459

I
iee .ini file variable UM-444
IEEE libraries UM-55
IEEE Std 1076 UM-20
IEEE Std 1364 UM-20, UM-81
IgnoreError .ini file variable UM-448, UM-455
IgnoreFailure .ini file variable UM-448, UM-455
IgnoreNote .ini file variable UM-448, UM-455
IgnoreVitalErrors .ini file variable UM-445
IgnoreWarning .ini file variable UM-448, UM-455
implicit operator, hiding with vcom -explicit CR-257
importing FPGA libraries UM-57
+incdir+ CR-290
incremental compilation
  automatic UM-84
  manual UM-84
    with Verilog UM-83
index checking UM-61
indexing signals, memories and nets CR-15
$init_signal_driver UM-368
init_signal_driver UM-359
$init_signal_spuy UM-371
init_signal_spuy UM-75, UM-362
init_userfs function UM-122
initial dialog box, turning on/off UM-440
initialization sequence UM-484
installation, license file, locating UM-474
instantiation in mixed-language design
  Verilog from VHDL UM-149
  VHDL from Verilog UM-152
instantiation label UM-238
interconnect delays CR-302, UM-388
internal signals, adding to a VCD file CR-233
iteration_limit, infinite zero-delay loops UM-500
IterationLimit .ini file variable UM-448, UM-455

K

keyboard shortcuts
List window UM-218, UM-460
Main window UM-183, UM-462
Source window UM-462
Wave window UM-274, UM-459

L

language templates UM-307
lecho command CR-163
left command CR-164
libraries
64-bit and 32-bit in same library UM-56
design libraries, creating CR-287, UM-49
design library types UM-48
design units UM-48
group use, setting up UM-493
IEEE UM-55
importing FPGA libraries UM-57
including precompiled modules UM-293
listing contents CR-259
mapping
from the command line UM-52
from the GUI UM-52
hierarchically UM-452
search rules UM-53
modelsim_lib UM-74
moving UM-53
naming UM-52
precompiled modules, including CR-290
predefined UM-54
refreshing library images CR-256, CR-293, UM-55
resource libraries UM-48
std library UM-54
Synopsys UM-55
vendor supplied, compatibility of CR-259
Verilog CR-307, UM-85, UM-146
VHDL library clause UM-54
working libraries UM-48
working with contents of UM-50
library simulator state variable UM-456
License variable in .ini file UM-448
licensing
feature name descriptions UM-476
license file, locating UM-474
License variable in .ini file UM-448
using the FLEXlm license manager UM-473
List window UM-204
adding items to CR-48
waveform comparison UM-354
see also windows, List window
LM_LICENSE_FILE environment variable UM-441
Imdown license server utility UM-478
Imgrd license server utility UM-478
Imremove license server utility UM-479
Imreread license server utility UM-479
Imstat license server utility UM-478
Imutil license server utility UM-479
Locate
specific time in Wave window UM-169
time cursors in Wave window UM-169
time markers in List window UM-169
location maps, referencing source files UM-501
lock message UM-470
locked memory
under HP-UX 10.2 UM-503
under Solaris UM-504
LockedMemory .ini file variable UM-449
log command CR-166
log file
log command CR-166
nolog command CR-174
overview UM-153
QuickSim II format CR-319
redirecting with -l CR-301
virtual log command CR-274
virtual nolog command CR-277
see also WLF files
Logic Modeling
SmartModel
command channel UM-408
SmartModel Windows
lmcmdwin commands UM-409
memory arrays UM-410
LSF
app note on using with ModelSim UM-24
lsifshift command CR-168
lsifsublist command CR-169

M

macro_option command CR-170
MacroNestingLevel simulator state variable UM-456
macros (DO files) UM-435
breakpoints, executing at CR-69
creating from a saved transcript UM-174
depth of nesting, simulator state variable UM-456
error handling UM-437
executing CR-138
forcing signals, nets, or registers CR-156
parameters
as a simulator state variable (n) UM-456
passing CR-138, UM-435
total number passed UM-456
relative directories CR-138
shifting parameter values CR-217
startup macros UM-452
.main clear command CR-37
Main window UM-173

see also windows, Main window
mapping libraries
from the command line UM-52
hierarchically UM-452
symbols
Dataflow window UM-202
Verilog states in mixed designs UM-147
math_complex package UM-55
math_real package UM-55
+maxdelays CR-291
.mc_scan_plusargs()
using with an elaboration file UM-64, UM-109
.mc_scan_plusargs, PLI routine CR-308
memory
enabling shared memory on Sun/Solaris UM-504
locking under HP-UX 10.2 UM-503
modeling in VHDL UM-507
menus
customizing UM-170
Dataflow window UM-187
List window UM-206
Main window UM-175
Process window UM-220
Signals window UM-223
Source window UM-230
Structure window UM-238
tearing off or pinning menus UM-170
Variables window UM-243
Wave window UM-249
messages UM-465
bad magic number UM-155
echoing CR-143
exit codes UM-468
getting more information CR-260, UM-466
lock message UM-470
ModelSim message system UM-466
redirecting UM-449
suppressing warnings from arithmetic packages UM-453
Tcl_init error UM-470
too few port connections UM-471
turning off assertion messages UM-452
warning, suppressing UM-467
MGC_LOCATION_MAP variable UM-441
+mindelays CR-291
miss and exclusion details UM-331
mixed-language simulation UM-143
access limitations UM-144
mnemonics, assigning to signal values CR-285
MODEL_TECH environment variable UM-441
MODEL_TECH_TCL environment variable UM-441
modeling memory in VHDL UM-507
ModelSim commands CR-27–CR-320
custom setup with daemon options UM-477
license file UM-474
modelsim command CR-171
MODELSIM environment variable UM-442
modelsim.ini found by ModelSim UM-484
default to VHDL93 UM-453
delay file opening with UM-453
environment variables in UM-451
force command default, setting UM-453
hierarchical library mapping UM-452
opening VHDL files UM-453
restart command defaults, setting UM-453
startup file, specifying with UM-452
transcript file created from UM-452
turning off arithmetic warnings UM-453
turning off assertion messages UM-452
modelsim.tcl file UM-454
modelsim_lib UM-74
path to UM-444
MODELSIM_TCL environment variable UM-442
Modified field, Project tab UM-36
mouse shortcuts
Main window UM-183, UM-462
Source window UM-462
Wave window UM-274, UM-459
MPF file, loading from the command line UM-46
mti_cosim_trace environment variable UM-442
MTI_TF_LIMIT environment variable UM-442
multiple drivers on unresolved signal UM-285
multiple simulations UM-153
multi-source interconnect delays CR-302
N

n simulator state variable UM-456
name case sensitivity, VHDL vs. Verilog CR-15
Name field
   Project tab UM-36
negative pulses
driving an error state CR-309
negative timing
   $setup/$recovery UM-116
   algorithm for calculating delays UM-96
check limits UM-96
   extending check limits CR-306
nets
adding to the Wave and List windows UM-226
Dataflow window, displaying in UM-186
drivers of, displaying CR-141
stimulus CR-156
values of
   displaying in Signals window UM-222
   examining CR-149
   forcing UM-225
   saving as binary log file UM-226
waveforms, viewing UM-246
new features UM-545
next and previous edges, finding UM-275, UM-460
next command CR-172
no space in time literal UM-285
NoCaseStaticError .ini file variable UM-445
NoDebug .ini file variable (VCOM) UM-445
NoDebug .ini file variable (VLOG) UM-446
noforce command CR-173
NoIndexCheck .ini file variable UM-445
+nolibcell CR-292
nolog command CR-174
non-blocking assignments UM-94
NoOthersStaticError .ini file variable UM-445
NoRangeCheck .ini file variable UM-445
notepad command CR-176
Notepad windows, text editing UM-183, UM-462
-notrigger argument UM-494
noview command CR-177
NoVital .ini file variable UM-445
NoVitalCheck .ini file variable UM-445
Now simulator state variable UM-456
now simulator state variable UM-456
+nowarn<CODE> CR-292
nowhen command CR-178
numeric_bit package UM-55
numeric_std package UM-55
NumericStdNoWarnings .ini file variable UM-449,

O

onbreak command CR-179
onElabError command CR-180
onerror command CR-181
operating systems supported UM-19
+opt UM-100
optimize for std_logic_1164 UM-286
Optimize_1164 .ini file variable UM-445
optimizing Verilog designs UM-99
design object visibility UM-105
   event order issues UM-107
gate-level UM-101
timing checks UM-107
without source UM-106
OptionFile entry in project files UM-287
order of events
   changing in Verilog CR-288
   in optimized designs UM-107
ordering files for compile UM-39
organizing projects with folders UM-43
others .ini file variable UM-445

P

packages
   examining constants in UM-497
   standard UM-54
textio UM-54
util UM-74
VITAL 1995 UM-71
VITAL 2000 UM-71
page setup
   Dataflow window UM-201
   Wave window UM-280
pan, Dataflow window UM-195
parameters
   making optional UM-436
   using with macros CR-138, UM-435
path delay mode UM-112
pathnames UM-350
   in VSM commands CR-14
   spaces in CR-13
PathSeparator .ini file variable UM-449, UM-455
pause command CR-182
PedanticErrors .ini file variable UM-445
performance
   enabling shared memory UM-504
improving for Verilog simulations UM-99
improving on HP-UX 10.2 UM-503
improving on Sun/Solaris UM-504

Performance Analyzer UM-319

%parent field UM-325
commands UM-327
getting started UM-321
hierarchical profile UM-322
in(%) field UM-324
interpreting data UM-322
name field UM-324
preferences, setting UM-327
profile report command UM-326
ranked profile UM-325
report option UM-326
results, viewing UM-322
statistical sampling UM-320
under(%) field UM-324
view_profile command UM-322
view_profile_ranked command UM-322

platforms, supported UM-19
play command CR-183
PLI
  specifying which apps to load UM-122
  Veriuser entry UM-122
PLI/VPI UM-121
  tracing UM-140
PLIOBJS environment variable UM-122, UM-442
popup
  toggling waveform popup on/off UM-266, UM-350
port driver data, capturing UM-400
ports, VHDL and Verilog UM-146
Postscript
  saving a waveform in UM-276
  saving the Dataflow display in UM-199
power add command CR-184
power report command CR-185
power reset command CR-186
precedence of variables UM-456
precision, simulator resolution UM-90, UM-144
pre-compiled libraries, optimizing with -fast UM-106
pref.tcl file UM-454
preference variables
  .ini files, located in UM-444
  code coverage UM-338
  editing UM-454
  Performance Analyzer UM-327
  saving UM-454
  set with Tcl set command UM-454
  simulator control UM-454
  Tcl files, located in UM-454

Waveform Compare UM-355
primitives, symbols in Dataflow window UM-202
printenv command CR-187
printing
  comparison differences UM-353
  Dataflow window display UM-199
  waveforms in the Wave window UM-276
Process window UM-219
  see also windows, Process window
processes
  values and pathnames in Variables window UM-242
  without wait statements UM-285
profile clear command CR-188
profile interval command CR-189
profile off command CR-190
profile on command CR-191
profile option command CR-192
profile report command CR-193, UM-326
profiler, see Performance Analyzer UM-319
Programming Language Interface UM-121
project tab
  information in UM-36
  sorting UM-37
projects UM-27
  accessing from the command line UM-46
  adding files to UM-31
  benefits UM-28
  compile order UM-39
    changing UM-39
    compiler options in UM-45
    compiling files UM-34
    context menu UM-37
    creating UM-30
    creating simulation configurations UM-41
    differences with earlier versions UM-29
    folders in UM-43
    grouping files in UM-40
    loading a design UM-35
MODELSIM environment variable UM-442
override mapping for work directory with vcom CR-256
  override mapping for work directory with vlog CR-294
overview UM-28
propagation, preventing X propagation CR-302
property list command CR-195
property wave command CR-196
'protect compiler directive CR-254, CR-292, UM-492
pulse error state CR-309
pwd command CR-197
Q

QuickSim II logfile format CR-319
Quiet .ini file variable
   VCOM UM-445
   VLOG UM-446
quietly command CR-198
quit command CR-199

R

race condition, problems with event order UM-92
radix
   changing in Signals, Variables, Dataflow, List, and Wave windows CR-200
   displaying character strings CR-285
   of signals being examined CR-150
   of signals in Wave window CR-59
   specifying in List window UM-209
radix command CR-200
range checking UM-61
   disabling CR-254
   enabling CR-255
ranked profile UM-325
readers and drivers UM-193
real type, converting to time UM-77
rebuilding supplied libraries UM-55
reconstruct RTL-level design busses UM-162
record command CR-201
records, values of, changing UM-242
$recovery UM-116
redirecting messages, TranscriptFile UM-449
reference region UM-346
reference signals UM-340
refreshing library images CR-256, CR-293, UM-55
register variables
   adding to the Wave and List windows UM-226
   values of
      displaying in Signals window UM-222
      saving as binary log file UM-226
      waveforms, viewing UM-246
report
   simulator control UM-440
   simulator state UM-440
report command CR-202
reporting
   compile history UM-37
   variable settings CR-17
RequireConfigForAllDefaultBinding variable UM-445
resolution
mixed designs UM-144
returning as a real UM-74
specifying with -t argument CR-303
verilog simulation UM-90
VHDL simulation UM-62
Resolution .ini file variable UM-449
resolution simulator state variable UM-456
resource library UM-48
restart command CR-204
defaults UM-453
   in GUI UM-177
toolbar button UM-181, UM-234, UM-255
restore command CR-206
restoring defaults UM-441
results, saving simulations UM-153
resume command CR-207
right command CR-208
RTL-level design busses
   reconstructing UM-162
run command CR-210
RunLength .ini file variable UM-449, UM-455

S

saving
   simulation options in a project UM-41
   Waveform Comparison differences UM-353
   waveforms UM-153
ScalarOpts .ini file variable UM-445, UM-446
scope, setting region environment CR-148
SDF
   errors and warnings UM-379
   instance specification UM-378
   interconnect delays UM-388
   mixed VHDL and Verilog designs UM-388
   specification with the GUI UM-379
troubleshooting UM-389
Verilog
   $sdf_annotate system task UM-382
   optional conditions UM-387
   optional edge specifications UM-386
   rounded timing values UM-387
   SDF to Verilog construct matching UM-383
VHDL
   resolving errors UM-381
   SDF to VHDL generic matching UM-380
search command CR-212
search libraries CR-307, UM-293
searching
   binary signal values in the GUI CR-21
in the source window UM-235
in the Structure window UM-241
List window
  signal values, transitions, and names CR-18, CR-139, CR-231, UM-213
  next and previous edge in Wave window CR-164, CR-208
values and names UM-169
Verilog libraries UM-85, UM-152
Wave window
  signal values, edges and names CR-164, CR-208, UM-266
searchlog command CR-214
seetime command CR-216
$setuphold UM-116
shared memory
  improving performance on HP-UX 10.2 UM-503
  improving performance on Sun/Solaris UM-504
shared objects
  loading FLI applications
    see ModelSim FLI Reference manual
  loading PLI/VPI C applications UM-124
  loading PLI/VPI C++ applications UM-127
shift command CR-217
Shortcuts
  text editing UM-183, UM-462
shortcuts
  command history CR-11, UM-461
  command line caveat CR-11, UM-461
  List window UM-218, UM-460
  Main window UM-462
  Main windows UM-183
  Source window UM-462
  Wave window UM-274, UM-459
show command CR-218
show differences UM-352
Show Drivers
  Dataflow window UM-193
show drivers
  Wave window UM-260
show source lines with errors UM-285
Show_source.ini file variable
  VCOM UM-446
  VLOG UM-446
Show_VitalChecksWarning.ini file variable UM-446
Show_Warning1.ini file variable UM-446
Show_Warning2.ini file variable UM-446
Show_Warning3.ini file variable UM-446
Show_Warning4.ini file variable UM-446
Show_Warning5.ini file variable UM-446
Signal Spy UM-75, UM-362
  overview UM-358
$signal_force UM-373
signal_force UM-75, UM-364
$signal_release UM-375
signal_release UM-75, UM-366
signals
  adding to a WLF file UM-226
  adding to the Wave and List windows UM-226
  alternative names in the List window (-label) CR-49
  alternative names in the Wave window (-label) CR-58
  applying stimulus to UM-225
  arrays of, indexing CR-15
  attributes of, using in expressions CR-24
  breakpoints CR-314, UM-228
  combining into a user-defined bus CR-58, UM-210, UM-259
Dataflow window, displaying in UM-186
  drivers of, displaying CR-141
  driving in the hierarchy UM-359
  environment of, displaying CR-148
  finding CR-153
  force time, specifying CR-157
hierarchy
  driving in UM-359, UM-368
  referencing in UM-75, UM-362, UM-371
  releasing anywhere in UM-366
  releasing in UM-75, UM-375
log file, creating CR-166
names of, viewing without hierarchy UM-265
pathnames in VSIM commands CR-14
radix
  specifying for examine CR-150
  specifying in List window CR-49
  specifying in Wave window CR-59
sampling at a clock change UM-494
states of, displaying as mnemonics CR-285
stimulus CR-156
transitions, searching for UM-271
  types, selecting which to view UM-225
  unresolved, multiple drivers on UM-285
values of
  converting to strings UM-495
  displaying in Signals window UM-222
  examining CR-149
  forcing anywhere in the hierarchy UM-75, UM-364, UM-373
  replacing with text CR-285
  saving as binary log file UM-226
  waveforms, viewing UM-246
Signals window UM-222
see also windows, Signals window

simulating
batch mode UM-490
command-line mode UM-490
comparing simulations UM-153, UM-339
default run length UM-298
delays, specifying time units for CR-17
design unit, specifying CR-298
elaboration file UM-63, UM-108
graphic interface to UM-288
iteration limit UM-298
mixed Verilog and VHDL designs
  compilers UM-144
  libraries UM-144
  resolution limit in UM-144
  Verilog parameters UM-145
  Verilog state mapping UM-147
  VHDL and Verilog ports UM-146
  VHDL generics UM-145
optimizing Verilog performance CR-289
saving dataflow display as a Postscript file UM-199
saving options in a project UM-41
saving simulations CR-166, CR-304, UM-153, UM-493
saving waveform as a Postscript file UM-276
speeding-up with Performance Analyzer UM-319
stepping through a simulation CR-222
stimulus, applying to signals and nets UM-225
stopping simulation in batch mode CR-316
time resolution UM-289
Verilog UM-89
  delay modes UM-111
  hazard detection UM-95
  optimizing performance UM-99
  resolution limit UM-90
  XL compatible simulator options UM-98
VHDL UM-62
viewing results in List window UM-204
VITAL packages UM-73

simulation
  event order in UM-92
Simulation Configuration
  creating UM-41
simulations
  saving results CR-130, CR-131, UM-153
  saving results at intervals UM-159
simulator resolution
  mixed designs UM-144
  returning as a real UM-74
  Verilog UM-90
  VHDL UM-62
vsim -t argument CR-303
  when comparing datasets UM-154
simulator state variables UM-454
simulator version CR-304, CR-312
simultaneous events in Verilog
  changing order CR-288
sizetf callback function Verilog UM-134
sm_entity UM-405
SmartModels
  creating foreign architectures with sm_entity UM-405
  invoking SmartModel specific commands UM-408
  linking to UM-404
lmcwin commands UM-409
memory arrays UM-410
Verilog interface UM-411
VHDL interface UM-404
so, shared object file
  loading PLI/VPI C applications UM-124
  loading PLI/VPI C++ applications UM-127
software updates UM-25
software version UM-180
sorting
  HDL items in GUI windows UM-170
source code, security UM-492
source directory, setting from source window UM-230
source errors, locating during compilation UM-283
source files, referencing with location maps UM-501
source libraries
  arguments supporting UM-86
  source lines with errors
    showing UM-285
Source window UM-229
see also windows, Source window
spaces in pathnames CR-13
specify path delays CR-309
speeding-up the simulation UM-319
splitio command CR-220
stability checking
  disabling CR-80
  enabling CR-81
Standard Developer’s Kit User Manual UM-24
standards supported UM-20
startup
  alternate to startup.do (vsim -do) CR-299
  environment variables access during UM-483
  files accessed during UM-482
  macro in the modelsim.ini file UM-449
  macros UM-452
  startup macro in command-line mode UM-491
  using a startup file UM-452
Startup .ini file variable UM-449
state variables UM-456
status bar
   Main window UM-183
status command CR-221
Status field
   Project tab UM-36
std .ini file variable UM-444
std_developerskit .ini file variable UM-444
Std_logic
   mapping to binary radix CR-21
std_logic_arith package UM-55
std_logic_signed package UM-55
std_logic_textio UM-55
std_logic_unsigned package UM-55
StdArithNoWarnings .ini file variable UM-449, UM-455
STDOUT environment variable UM-442
step command CR-222
stimulus
   applying to signals and nets UM-225
   modifying for elaboration file UM-64, UM-109
stop command CR-223
Structure window UM-237
   see also windows, Structure window
support UM-25
symbol mapping
   Dataflow window UM-202
symbolic link to design libraries (UNIX) UM-53
Synopsis hardware modeler UM-414
synopsys .ini file variable UM-444
Synopsys libraries UM-55
synthesis
   rule compliance checking UM-285, UM-445
system calls
   VCD UM-392
   Verilog UM-113
system commands UM-427
system tasks
   VCD UM-392
   Verilog UM-113

T

tab stops, in the Source window UM-236
tb command CR-224
Tcl UM-419–UM-430
   command separator UM-426
   command substitution UM-425
   command syntax UM-422
evaluation order UM-426
history shortcuts CR-11, UM-461
Man Pages in Help menu UM-180
preference variables UM-454
relational expression evaluation UM-426
variable substitution UM-427
VSIM Tcl commands UM-428
Tcl commands
   set UM-454
   Tcl_init error message UM-470
tech notes UM-24
technical support UM-25
temp files, VSOUT UM-443
test region UM-346
test signals UM-340
testbench, accessing internal items from UM-357
text and command syntax UM-23
Text editing UM-183, UM-462
TextIO package
   alternative I/O files UM-70
   containing hexadecimal numbers UM-69
dangling pointers UM-69
ENDFILE function UM-69
ENDLINE function UM-69
file declaration UM-66
implementation issues UM-68
providing stimulus UM-70
standard input UM-67
standard output UM-67
WRITE procedure UM-68
WRITE_STRING procedure UM-68
TF routines UM-138
TFMPC
   disabling warning CR-308
   explanation UM-471
time
   absolute, using @ CR-17
   simulation time units CR-17
   time resolution as a simulator state variable UM-456
time literal, missing space UM-285
time resolution
   in mixed designs UM-144
   in Verilog UM-90
   in VHDL UM-62
   setting
      with the GUI UM-289
      with vsim command CR-303
time type, converting to real UM-76
time-based breakpoints UM-228
timescale directive warning, disabling CR-308
timing
$setuphold/$recovery UM-116
annotation UM-377
differences shown by comparison UM-350
disabling checks CR-292, CR-302
negative check limits
described UM-96
extending CR-306
TMPDIR environment variable UM-442
to_real VHDL function UM-76
to_time VHDL function UM-77
toggle add command CR-225
toggle checking UM-499
toggle report command CR-226
toggle reset command CR-227
toggle statistics
enabling CR-225
merging multiple output files UM-499
reporting CR-226
resetting CR-227
toggling waveform popup on/off UM-266, UM-350
tolerance
leading edge UM-348
trailing edge UM-348
too few port connections, explanation UM-471
toolbar
Dataflow window UM-190
Main window UM-181
Wave window UM-254
tooltip, toggling waveform popup UM-266
tracing
events UM-196
source of unknown UM-197
transcribe command CR-228
transcript
clearing CR-37
file name, specified in modelsim.ini UM-452
saving UM-174
TranscriptFile variable in .ini file UM-449
using as a DO file UM-174
transcript command CR-229
transcript file
redirecting with -l CR-301
transitions, signal, finding CR-164, CR-208
tree windows
VHDL and Verilog items in UM-171
viewing the design hierarchy UM-171
TreeUpdate command CR-324
triggers, in the List window, setting UM-212, UM-511
TScale, disabling warning CR-308
TSSI CR-329
in VCD files UM-400
tssi2mti command CR-230
type
converting real to time UM-77
converting time to real UM-76
Type field, Project tab UM-36

U
-u CR-294
unbound component UM-285
UnbufferedOutput .ini file variable UM-449
unit delay mode UM-112
unknowns, tracing UM-197
unresolved signals, multiple drivers on UM-285
up command CR-231
UpCase .ini file variable UM-446
updates UM-25
use 1076-1993 language standard UM-284
use clause, specifying a library UM-54
use explicit declarations only UM-285
user hook Tcl variable UM-310
user-defined bus CR-58, UM-161, UM-210, UM-259
UserTimeUnit .ini file variable UM-450, UM-455
util package UM-74

V
-v CR-294
values
describe HDL items CR-134
examine HDL item values CR-149
of HDL items UM-235
replacing signal values with strings CR-285
variable settings report CR-17
variables
environment variables UM-441
LM_LICENSE_FILE UM-441
personal preferences UM-440
precedence between .ini and .tcl UM-456
reading from the .ini file UM-451
setting environment variables UM-441
simulator control UM-454
simulator state variables
current settings report UM-440
iteration number UM-456
name of entity or module as a variable UM-456
resolution UM-456
simulation time UM-456
Variables window UM-242
see also windows, Variables window
variables, HDL
  describing CR-134
  value of
    changing from command line CR-72
    changing with the GUI UM-242
  examining CR-149
variables, Tcl, user hook UM-310
vcd add command CR-233
vcd checkpoint command CR-234
vcd comment command CR-235
vcd dumpports command CR-236
vcd dumpportslimit command CR-240
vcd dumpportsoff command CR-241
vcd dumpportsall command CR-238
vcd dumpportson command CR-242
vcd file command CR-243

VCD files UM-391
  adding items to the file CR-233
  capturing port driver data CR-236, UM-400
  case sensitivity UM-394
  converting to WLF files CR-251
  creating CR-233, UM-394
  dumping variable values CR-234
  dumpports tasks UM-392
  flushing the buffer contents CR-247
  from VHDL source to VCD output UM-397
  inserting comments CR-235
  internal signals, adding CR-233
  specifying maximum file size CR-248
  specifying name of CR-245
  specifying the file name CR-243
  state mapping CR-243, CR-245
  supported TSSI states UM-400
  turn off VCD dumping CR-249
  turn on VCD dumping CR-250
  VCD system tasks UM-392
  viewing files from another tool CR-251
vcd files command CR-245
vcd flush command CR-247
vcd limit command CR-248
vcd off command CR-249
vcd on command CR-250
vcd2wlf command CR-251
vcom command CR-252
vdel command CR-258
vdir command CR-259
vector elements, initializing CR-72
vendor libraries, compatibility of CR-259
Vera, see Vera documentation

Verilog
  ACC routines UM-137
  capturing port driver data with -dumpports CR-243, UM-400
  cell libraries UM-111
  compiler directives UM-119
  compiling and linking PLI C applications UM-124
  compiling and linking PLI C++ applications UM-127
  compiling design units UM-82
  compiling with XL ’uselib compiler directive UM-87
  component declaration UM-150
  creating a design library UM-82
  event order in simulation UM-92
  instantiation criteria in mixed-language design UM-149
  instantiation of VHDL design units UM-152
  language templates UM-307
  library usage UM-85
  mapping states in mixed designs UM-147
  mixed designs with VHDL UM-143
  parameters UM-145
  SDF annotation UM-382
  sdf_annotate system task UM-382
  simulating UM-89
    delay modes UM-111
    XL compatible options UM-98
  simulation hazard detection UM-95
  simulation resolution limit UM-90
  SmartModel interface UM-411
  source code viewing UM-229
  standards UM-20
  system tasks UM-113
  TF routines UM-138
  XL compatible compiler options UM-86
  XL compatible routines UM-140
  XL compatible system tasks UM-116
  verilog.ini file variable UM-444

Verilog 2001, current implementation UM-20, UM-81

Verilog PLI/VPI UM-121–UM-142
  64-bit support in the PLI UM-140
  compiling and linking PLI/VPI C applications UM-124
  compiling and linking PLI/VPI C++ applications UM-127
  debuggin PLI/VPI code UM-140
  PLI callback reason argument UM-133
  PLI support for VHDL objects UM-136
  registering PLI applications UM-121
  registering VPI applications UM-123
  specifying the PLI/VPI file to load UM-130
Veriuser .ini file variable UM-122, UM-450
Veriuser, specifying PLI applications UM-122
veriuser.c file UM-135
verror command CR-260
version
  obtaining via Help menu UM-180
  obtaining with vsim command CR-304
  obtaining with vsim<info> commands CR-312
vgencomp command CR-261
VHDL
  compiling design units UM-61
  creating a design library UM-61
  delay file opening UM-453
  dependency checking UM-61
  field naming syntax CR-15
  file opening delay UM-453
  foreign language interface UM-78
  hardware model interface UM-414
  instantiation from Verilog UM-152
  instantiation of Verilog UM-145
  language templates UM-307
  library clause UM-54
  mixed designs with Verilog UM-143
  object support in PLI UM-136
  simulating UM-62
  SmartModel interface UM-404
  source code viewing UM-229
  standards UM-20
  VITAL package UM-55
VHDL utilities UM-74, UM-75, UM-362, UM-371
  get_resolution() UM-74
  to_real() UM-76
  to_time() UM-77
VHDL93 .ini file variable UM-446
view command CR-263
view_profile command UM-322
view_profile_ranked command UM-322
viewing
  design hierarchy UM-171
  library contents UM-50
  waveforms CR-304, UM-153
virtual count commands CR-265
virtual define command CR-266
virtual delete command CR-267
virtual describe command CR-268
virtual expand commands CR-269
virtual function command CR-270
virtual hide command CR-273, UM-162
virtual log command CR-274
virtual nohide command CR-276
virtual nolog command CR-277
virtual objects UM-161
  virtual functions UM-162
  virtual regions UM-163
  virtual signals UM-161
  virtual types UM-163
virtual region command CR-279, UM-163
virtual regions
  reconstruct the RTL hierarchy in gate-level design UM-163
virtual save command CR-280, UM-162
virtual show command CR-281
virtual signal command CR-282, UM-161
virtual signals
  reconstruct RTL-level design busses UM-162
  reconstruct the original RTL hierarchy UM-161
  virtual hide command UM-162
virtual type command CR-285
VITAL
  compiling and simulating with accelerated VITAL packages UM-73
  compliance warnings UM-72
  specification and source code UM-71
  VITAL packages UM-71
vital95 .ini file variable UM-444
vlib command CR-287
vlog command CR-288
vlog.opt file UM-287
vmake command CR-296
vmap command CR-297
VPI, registering applications UM-123
VPI/PLI UM-121
  compiling and linking C applications UM-124
  compiling and linking C++ applications UM-127
vsim build date and version CR-312
vsim command CR-298
VSOUT temp file UM-443

W

warnings
  exit codes UM-468
  getting more information UM-466
  suppressing VCOM warning messages CR-255, UM-467
  suppressing VLOG warning messages CR-292, UM-467
  suppressing VSIM warning messages CR-308, UM-467
too few port connections UM-471
turning off warnings from arithmetic packages UM-
wave format file  UM-248
wave log format (WLF) file  CR-304, UM-153
of binary signal values  CR-166
see also WLF files
wave viewer, Dataflow window  UM-194
Wave window  UM-246
    compare waveforms  UM-350
    in the Dataflow window  UM-194
    toggling waveform popup on/off  UM-266, UM-350
values column  UM-351
see also windows, Wave window
wave, adding  CR-57
    .wave.tree interrupt command  CR-38
    .wave.tree zoomfull command  CR-39
    .wave.tree zoomin command  CR-40
    .wave.tree zoomlast command  CR-41
    .wave.tree zoomout command  CR-42
    .wave.tree zoomrange command  CR-43
WaveActivateNextPane command  CR-324
Waveform Comparison  CR-83, UM-339
    add region  UM-346
    adding signals  UM-345
    clear differences  UM-352
    clocked comparison  UM-341, UM-347
    compare by region  UM-346
    compare by signal  UM-345
    compare commands  UM-355
    compare menu  UM-352
    compare options  UM-349
    compare tab  UM-344
    comparison method tab  UM-347
    comparison modes  UM-341
    comparison wizard  UM-352
    continuous comparison  UM-341, UM-348
dataset  UM-340
    dataset, specifying  UM-343
    difference markers  UM-350
    end  UM-352
    features  UM-340
    flattened designs  UM-342
    graphic interface  UM-343
    hierarchical designs  UM-342
    icons  UM-351
    introduction  UM-340
    leading edge tolerance  UM-348
    limit count  UM-349
    List window display  UM-354
    pathnames  UM-350
    preference variables  UM-355
    printing differences  UM-353
    reference dataset  UM-343
    reference region  UM-346
    reference signals  UM-340
    reload  UM-353
    rules  UM-353
    run comparison  UM-352
    save differences  UM-353
    show differences  UM-352
    specify when expression  UM-347, UM-348
    start  UM-352
    Tcl preference variables  UM-355
    test dataset  UM-343
    test region  UM-346
    test signals  UM-340
    timing differences  UM-350
    tolerances  UM-341
    trailing edge tolerance  UM-348
    values column  UM-351
    Verilog matching  UM-349
    VHDL matching  UM-349
    Wave window display  UM-350
    when statement  UM-347
    write report  UM-353
waveform logfile
    log command  CR-166
    overview  UM-153
see also WLF files
waveform popup  UM-266, UM-350
waveforms  UM-153
    halting drawing  CR-38
    saving and viewing  CR-166, UM-154
    saving and viewing in batch mode  UM-493
    viewing  UM-246
WaveRestoreCursors command  CR-324
WaveRestoreZoom command  CR-324
WaveSignalNameWidth .ini file variable  UM-450
welcome dialog, turning on/off  UM-440
when command  CR-314
when statement
    setting signal breakpoints  UM-228
    specifying for waveform comparison  UM-347
    time-based breakpoints  CR-317
where command  CR-318
wildcard characters
    for pattern matching in simulator commands  CR-16
Windows
    Main window
        text editing  UM-183, UM-462
    Source window
        text editing  UM-183, UM-462
windows
buttons, adding to UM-310
coverage_summary UM-330
Dataflow window UM-186
  toolbar UM-190
  zooming UM-195
finding HDL item names in UM-169
List window UM-204
  adding HDL items UM-205
  adding signals with a WLF file UM-226
  formatting HDL items UM-208
  output file CR-325
  saving data to a file UM-217
  saving the format of CR-323
  setting triggers UM-212, UM-511
  time markers UM-169
Main window UM-173
  adding user-defined buttons CR-45
  status bar UM-183
  time and delta display UM-183
  toolbar UM-181
opening
  from command line CR-263
  multiple copies UM-170
  with the GUI UM-176
Process window UM-219
  displaying active processes UM-219
  specifying next process to be executed UM-219
  viewing processing in the region UM-219
saving position and size UM-170
searching for HDL item values in UM-169
Signals window UM-222
  VHDL and Verilog items viewed in UM-222
Source window UM-229
  setting tab stops UM-236
  viewing HDL source code UM-229
Structure window UM-237
  instance names UM-238
  selecting items to view in Signals window UM-222
  VHDL and Verilog items viewed in UM-237
  viewing design hierarchy UM-237
Variables window UM-242
  VHDL and Verilog items viewed in UM-242
Wave window UM-246
  adding HDL items to UM-248
  adding signals with a WLF file UM-226
  cursor measurements UM-270
  display properties UM-265
  display range (zoom), changing UM-271
  format file, saving UM-248
  path elements, changing CR-112, UM-450
  searching for HDL item values UM-267
  time cursors UM-269
  zooming UM-271
WLF files
  adding items to UM-226
  comparing UM-340
  creating from VCD CR-251
  limiting size CR-304
  log command CR-166
  overview UM-154
  repairing CR-321
  saving CR-130, CR-131, UM-155
  saving at intervals UM-159
  specifying name CR-304
  using in batch mode UM-493
wlf2log command CR-319
wlfrecover command CR-321
Work library UM-48
workspace UM-173
write cell_report command CR-322
write format command CR-323
write list command CR-325
write preferences command CR-326
write report command CR-327
write transcript command CR-328
write tssi command CR-329
write wave command CR-331
write, waveform comparison report UM-353

X

X

tracing unknowns UM-197
X propagation, preventing CR-302

Y

-y CR-294

Z

zero delay elements UM-513
zero delay mode UM-112
zero-delay loop, infinite UM-500
zero-delay oscillation UM-500
zero-delay race condition UM-92
zoom
  Dataflow window UM-195
from Wave toolbar buttons UM-271
saving range with bookmarks UM-272
with the mouse UM-272