# Library Compiler™ Variables and Attributes

Version O-2018.06, June 2018

SYNOPSYS®

# Copyright Notice and Proprietary Information



## Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

## Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

## Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

## Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

## Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

## Copyright Notice for the Command-Line Editing Feature



## Copyright Notice for the Line-Editing Library



## Copyright Notice for the Nonlinear Optimization Library

**Contents**

# lc_check_lib_keep_line_number

Specifies to keep line numbers for DB objects in read_lib so check_library will report results with line numbers in LIBCHK-3xx tables.

## TYPE

boolean

## DEFAULT

false

## DESCRIPTION

When the tcl variable is set to true, the line numbers of the library objects or groups will be kept in memory in read_lib so check_library will report results with line numbers in LIBCHK-3xx tables. This variable has to be set before read_lib, and it does not work with read_db. As a side benefit, when the line numbers are kept, for buses check_library will perform the checks on the buses as a whole instead of bit blasted pins, resulting in better performance and shorter check reports without duplications on each bit blasted pins.

## EXAMPLE

The following example sets the value of the **lc_check_lib_keep_line_number** variable:

```
prompt> set lc_check_lib_keep_line_number true
```

## SEE ALSO

`check_library(2)`

# lc_enable_10nm_mode

Specifies 10 nm mode for library checking/screening.

## TYPE

boolean

## DEFAULT

false

## DESCRIPTION

When the tcl variable is set to true, the 10nm default tolerances for constraint and voltage_range are used and the interpolation and constraint table monotonicity analysis are included in **set_check_library_options -char_integrity**. The 10nm default tolerances are Relative tolerance for constraint : 0.01 Absolute tolerance for constraint : 0.001ns Relative tolerance for capacitance : 0.01 Absolute tolerance for capacitance : 0.001pF Relative tolerance for voltage : 0.01 Absolute tolerance for voltage : 0.01V

## EXAMPLE

The following example sets the value of the **lc_enable_10nm_mode** variable:

```
prompt> set lc_enable_10nm_mode true
```

## SEE ALSO

set_check_library_options(2)

# plot_table

plot lookup table(s) by gnuplot.

## SYNTAX

```
plot_table
   [-aux equation_list]
   [-aux2 equation_list]
   [-derive_output_voltage]
   [-derive_output_voltage_equation]
   [-initv1 Vss(Vdd)]
   [-index {variable1 index1 variables2 index2}|index1_id:index2_id]
   [-slice {variable index}|variable_id:index_id]
   [-title title]
   [-xlabel xlabel]
   [-ylabel ylabel]
   [-y2label y2label]
   [-legend legend_list]
   [-save png_file_path]
   [-clear]
   [-dump data|script|all]
   [object_list]
```

### Data Types

| | |
|---|---|
| *equation_list* | list of strings |
| *equation* | string |
| *Vss(Vdd)* | float |
| *{variable1 index1 variables2 index2}\|index1_id:index2_id* | list or int:int |
| *{variable index}\|variable_id:index_id* | list or int:int |
| *title* | string |
| *xlabel* | string |
| *ylabel* | string |
| *y2label* | string |
| *legend_list* | list of strings |
| *png_file_path* | string |
| *data\|script\|all* | string |
| *object_list* | list of TCL variables |

## ARGUMENTS

**-aux equation_list**

Specifies auxiliary curves or surfaces by equations.

**-aux2 equation_list**

Specifies auxiliary curves aligning to 2nd Y-Axis for CCS object only.

**-derive_output_voltage**

Add output voltage curve which is derived upon equation: V1+0.5*(I2+I1)*(T2-T1)/C.

**-derive_output_voltage_equation**

Customize the equation to derive output voltage curve.

**-initv1 Vss(Vdd)**

Specifies Vss(Vdd) as initial V1 for derived voltage. 0 will be used as default.

**-index {variable1 index1 variables2 index2}|index1_id:index2_id**

Specifies the grid points for CCS only. Note that index1_id and/or index2_id is 0-based.

**-slice {variable index}|variable_id:index_id**

Specifies the slice to plot for non-CCS 2D object. Note that variable_id and index_id is 0-based.

**-title title**

Specifies title for the plot chart.

**-xlabel xlabel**

Specifies label of x-axis for the plot chart.

**-ylabel ylabel**

Specifies label of y-axis for the plot chart.

**-y2label y2label**

Specifies label of y2-axis for the plot chart.

**-legend legend_list**

Specifies legends for lookup table.

**-save png_file_path**

Specifies the name of image file to be saved in png format.

**-clear**

Close all opened terminals and remove all intermediate data.

**-dump**

Print out data and/or scripts generated, instead of plot them.

**object_list**

Specifies lookup table object variables.

# DESCRIPTION

This command plots lookup table object(s) given by $object_list. One of the following types can be plotted:

1. single or multiple lookup tables in 2D/3D data chart according to tables' dimension(1/2)
2. single CCS object in overview mode, that is, all vectors of the CCS object being plotted side by si
3. single CCS object of the specified vector(point)

One or more lookup table(s) with same dimension(1 or 2) can be plotted in one window.

# EXAMPLES

Several cases are provided here for example:

```
Case 1 - plot curve (i.e. 1d data) with customized title:
lc_shell>set lc_enable_plot_table true
lc_shell>read_lib case1_fall_power.lib
lc_shell>set pin [get_lib_pins library1/cell/pin*]
lc_shell>set internal_power [get_lib_objects -of_objects $pin -class_type internal_power]
lc_shell>set luts [get_lookup_table -of_objects $internal_power fall_power]
lc_shell>set lut [index_collection $luts 0]
lc_shell>plot_table $lut -title "cell(cell1) pin(pin1) internal_power"


Case 2 - plot surface (i.e. 2d data), also save to image file:
lc_shell>set lc_enable_plot_table true
lc_shell>read_lib case2_cell_rise.lib
lc_shell>set pin [get_lib_pins library2/cell/pin*]
lc_shell>set timing [get_lib_objects -of_objects $pin -class_type timing]
lc_shell>set luts [get_lookup_table -of_objects $timing cell_rise]
lc_shell>set lut [index_collection $luts 0]
lc_shell>plot_table $lut -save "cell_rise.png"


Case 3 - plot more than 1 objects, with customized legends:
lc_shell>set lc_enable_plot_table true
lc_shell>read_lib case3_cell_rise_cell_fall.lib
lc_shell>set pin [get_lib_pins library3/cell/pin*]
lc_shell>set timing [get_lib_objects -of_objects $pin -class_type timing]
lc_shell>set luts_rise [get_lookup_table -of_objects $timing cell_rise]
lc_shell>set luts_fall [get_lookup_table -of_objects $timing cell_fall]
lc_shell>set lut_rise [index_collection $luts_rise 0]
lc_shell>set lut_fall [index_collection $luts_fall 0]
lc_shell>plot_table [list $lut_rise $lut_fall] -legend [list cell_rise cell_fall]


Case 4 - plot all vectors/1 vector for CCS object:
lc_shell>set lc_enable_plot_table true
lc_shell>read_lib case4_ccs.lib
lc_shell>set pin [get_lib_pins library4/cell/pin*]
lc_shell>set timing [get_lib_objects -of_objects $pin -class_type timing]
lc_shell>set luts [get_lookup_table -of_objects $timing output_current_rise]
```

```
lc_shell>set lut [index_collection $luts 0]
lc_shell>plot_table $lut -derive_output_voltage
lc_shell>plot_table $lut -index 7:3 -derive_output_voltage

Case 4.1 - no derive_output_voltage:
lc_shell>plot_table $lut -index 7:3

Case 4.2 - customized equation to derive output voltage curve, with auxiliary line aligning to 2nd Y-A
lc_shell>plot_table $lut -index 7:3 -derive_output_voltage -derive_output_voltage_equation V1+0.5*(abs


Case 5 - plot slices for non-CCS 2D object:
lc_shell>set lc_enable_plot_table true
lc_shell>read_lib case5_dcu.lib
lc_shell>set Cell_dcu [get_lookup_table -of_objects [get_lib_cells */*]  dc_current]
lc_shell>set lutdcu [index_collection $Cell_dcu 0]
lc_shell>set vIdx [lookup_table index $Cell_dcu]
lc_shell>set nInputVoltageIndexCount [llength [lindex $vIdx 0]]
lc_shell>for { set i 0 } { $i<$nInputVoltageIndexCount } { incr i } {
lc_shell>  plot_table $lutdcu -slice 0:$i
lc_shell>}
```

## SEE ALSO

lookup_table(2)

# sh_allow_tcl_with_set_app_var

Allows the **set_app_var** and **get_app_var** commands to work with application variables.

## TYPE

string

## DEFAULT

application specific

## DESCRIPTION

Normally the **get_app_var** and **set_app_var** commands only work for variables that have been registered as application variables. Setting this variable to **true** allows these commands to set a Tcl global variable instead.

These commands issue a CMD-104 error message for the Tcl global variable, unless the variable name is included in the list specified by the **sh_allow_tcl_with_set_app_var_no_message_list** variable.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var_no_message_list(2)
```

# sh_allow_tcl_with_set_app_var_no_message_l

Suppresses CMD-104 messages for variables in this list.

## TYPE

string

## DEFAULT

application specific

## DESCRIPTION

This variable is consulted before printing the CMD-104 error message, if the **sh_allow_tcl_with_set_app_var** variable is set to **true**. All variables in this Tcl list receive no message.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var(2)
```

# sh_arch

Indicates the system architecture of your machine.

## TYPE

string

## DEFAULT

platform-dependent

## DESCRIPTION

The **sh_arch** variable is set by the application to indicate the system architecture of your machine. Examples of machines being used are sparcOS5, amd64, and so on. This variable is read-only.

# sh_command_abbrev_mode

Sets the command abbreviation mode for interactive convenience.

## TYPE

string

## DEFAULT

application specific

## DESCRIPTION

This variable sets the command abbreviation mode as an interactive convenience. Script files should not use any command or option abbreviation, because these files are then susceptible to command changes in subsequent versions of the application.

Although the default value is **Anywhere**, it is recommended that the site startup file for the application set this variable to **Command-Line-Only**. It is also possible to set the value to **None**, which disables abbreviations altogether.

To determine the current value of this variable, use the **get_app_var sh_command_abbrev_mode** command.

## SEE ALSO

```
sh_command_abbrev_options(3)
get_app_var(2)
set_app_var(2)
```

# sh_command_abbrev_options

Turns off abbreviation of command dash option names when false.

## TYPE

boolean

## DEFAULT

application specific

## DESCRIPTION

When command abbreviation is currently off (see sh_command_abbrev_mode) then setting this variable to false will also not allow abbreviation of command dash options. This variable also impacts abbreviation of the values specified to command options that expect values to be one of an allowed list of values.

This variable exists to be backward compatible with previous tool releases which always allowed abbreviation of command dash options and option values regardless of the command abbreviation mode.

It is recommended to set the value of this variable to false.

To determine the current value of this variable, use the **get_app_var sh_command_abbrev_options** command.

## SEE ALSO

sh_command_abbrev_mode(3)
get_app_var(2)
set_app_var(2)

# sh_command_log_file

Specifies the name of the file to which the application logs the commands you executed during the session.

## TYPE

string

## DEFAULT

empty string

## DESCRIPTION

This variable specifies the name of the file to which the application logs the commands you run during the session. By default, the variable is set to an empty string, indicating that the application's default command log file name is to be be used. If a file named by the default command log file name cannot be opened (for example, if it has been set to read only access), then no logging occurs during the session.

This variable can be set at any time. If the value for the log file name is invalid, the variable is not set, and the current log file persists.

To determine the current value of this variable, use the **get_app_var sh_command_log_file** command.

## SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

# sh_continue_on_error

Allows processing to continue when errors occur during script execution with the **source** command.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

This variable is deprecated. It is recommended to use the **-continue_on_error** option to the **source** command instead of this variable because that option only applies to a single script, and not the entire application session.

When set to **true**, the **sh_continue_on_error** variable allows processing to continue when errors occur. Under normal circumstances, when executing a script with the **source** command, Tcl errors (syntax and semantic) cause the execution of the script to terminate.

When **sh_continue_on_error** is set to **false**, script execution can also terminate due to new error and warning messages based on the value of the **sh_script_stop_severity** variable.

To determine the current value of the **sh_continue_on_error** variable, use the **get_app_var sh_continue_on_error** command.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_script_stop_severity(3)
```

# sh_deprecated_is_error

Raise a Tcl error when a deprecated command is executed.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

When set this variable causes a Tcl error to be raised when an deprecated command is executed. Normally only a warning message is issued.

## SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

# sh_dev_null

Indicates the current null device.

## TYPE

string

## DEFAULT

platform dependent

## DESCRIPTION

This variable is set by the application to indicate the current null device. For example, on UNIX machines, the variable is set to **/dev/null**. This variable is read-only.

## SEE ALSO

`get_app_var(2)`

# sh_enable_page_mode

Displays long reports one page at a time (similar to the UNIX **more** command.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

This variable displays long reports one page at a time (similar to the UNIX **more** command), when set to **true**. Consult the man pages for the commands that generate reports to see if they are affected by this variable.

To determine the current value of this variable, use the **get_app_var sh_enable_page_mode** command.

## SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

# sh_enable_stdout_redirect

Allows the redirect command to capture output to the Tcl stdout channel.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

When set to **true**, this variable allows the redirect command to capture output sent to the Tcl stdout channel. By default, the Tcl **puts** command sends its output to the stdout channel.

## SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

# sh_help_shows_group_overview

Changes the behavior of the "help" command.

## TYPE

string

## DEFAULT

application specific

## DESCRIPTION

This variable changes the behavior of the **help** command when no arguments are specified to help. Normally when no arguments are specified an informational message with a list of available command groups is displayed.

When this variable is set to false the command groups and the commands in each group is printed instead. This variable exists for backward compatibility.

## SEE ALSO

`help(2)`
`set_app_var(2)`

# sh_new_variable_message

Controls a debugging feature for tracing the creation of new variables.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

The **sh_new_variable_message** variable controls a debugging feature for tracing the creation of new variables. Its primary debugging purpose is to catch the misspelling of an application-owned global variable. When set to **true**, an informational message (CMD-041) is displayed when a variable is defined for the first time at the command line. When set to **false**, no message is displayed.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var command man page for details.**

Other variables, in combination with **sh_new_variable_message**, enable tracing of new variables in scripts and Tcl procedures.

Warning: This feature has a significant negative impact on CPU performance when used with scripts and Tcl procedures. This feature should be used only when developing scripts or in interactive use. When you turn on the feature for scripts or Tcl procedures, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message** command.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message_in_proc(3)
sh_new_variable_message_in_script(3)
```

# sh_new_variable_message_in_proc

Controls a debugging feature for tracing the creation of new variables in a Tcl procedure.

## TYPE

Boolean

## DEFAULT

false

## DESCRIPTION

The **sh_new_variable_message_in_proc** variable controls a debugging feature for tracing the creation of new variables in a Tcl procedure. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. Please see the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. Enabling the feature simply enables the **print_proc_new_vars** command. In order to trace the creation of variables in a procedure, this command must be inserted into the procedure, typically as the last statement. When all of these steps have been taken, an informational message (CMD-041) is generated for new variables defined within the procedure, up to the point that the **print_proc_new_vars** commands is executed.

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_proc** command.

## SEE ALSO

```
get_app_var(2)
print_proc_new_vars(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_script(3)
```

# sh_new_variable_message_in_script

Controls a debugging feature for tracing the creation of new variables within a sourced script.

## TYPE

Boolean

## DEFAULT

false

## DESCRIPTION

The **sh_new_variable_message_in_script** variable controls a debugging feature for tracing the creation of new variables within a sourced script. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. In that case, an informational message (CMD-041) is displayed when a variable is defined for the first time. When **sh_new_variable_message_in_script** is set to **false** (the default), no message is displayed at the time that the variable is created. When the **source** command completes, however, you see messages for any new variables that were created in the script. This is because the state of the variables is sampled before and after the **source** command. It is not because of inter-command sampling within the script. So, this is actually a more efficient method to see if new variables were created in the script.

For example, given the following script a.tcl:

```
echo "Entering script"
set a 23
echo a = $a
```

```
set b 24
echo b = $b
echo "Exiting script"
```

When **sh_new_variable_message_in_script** is **false** (the default), you see the following when you source the script:

```
prompt> source a.tcl
Entering script
a = 23
b = 24
Exiting script
Information: Defining new variable 'a'. (CMD-041)
Information: Defining new variable 'b'. (CMD-041)
prompt>
```

Alternatively, when **sh_new_variable_message_in_script** is **true**, at much greater cost, you see the following when you source the script:

```
prompt> set sh_new_variable_message_in_script true
Warning: Enabled new variable message tracing -
         Tcl scripting optimization disabled. (CMD-042)
true
prompt> source a.tcl
Entering script
Information: Defining new variable 'a'. (CMD-041)
a = 23
Information: Defining new variable 'b'. (CMD-041)
b = 24
Exiting script
prompt>
```

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_script** command.

# SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_proc(3)
```

# sh_obsolete_is_error

Raise a Tcl error when an obsolete command is executed.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

When set this variable causes a Tcl error to be raised when an obsolete command is executed. Normally only a warning message is issued.

Obsolete commands have no effect.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

# sh_product_version

Indicates the version of the application currently running.

## TYPE

string

## DESCRIPTION

This variable is set to the version of the application currently running. The variable is read only.

To determine the current value of this variable, use the **get_app_var sh_product_version** command.

## SEE ALSO

`get_app_var(2)`

# sh_script_stop_severity

Indicates the error message severity level that would cause a script to stop running before it completes.

## TYPE

string

## DEFAULT

application specific

## DESCRIPTION

When a script is run with the **source** command, there are several ways to get it to stop running before it completes. One is to use the **sh_script_stop_severity** variable. This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a script to stop.
- When set to **W**, the generation of one or more warning or error messages causes a script to stop.
- When set to **none**, the generation messages does not cause the script to stop.

Note that **sh_script_stop_severity** is ignored if **sh_continue_on_error** is set to **true**.

To determine the current value of this variable, use the **get_app_var sh_script_stop_severity** command.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
```

# sh_source_emits_line_numbers

Indicates the error message severity level that causes an informational message to be issued, listing the script name and line number where that message occurred.

## TYPE

string

## DEFAULT

application specific

## DESCRIPTION

When a script is executed with the **source** command, error and warning messages can be emitted from any command within the script. Using the **sh_source_emits_line_numbers** variable, you can help isolate where errors and warnings are occurring.

This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a CMD-082 informational message to be issued when the command completes, giving the name of the script and the line number of the command.
- When set to **W**, the generation of one or more warning or error messages causes a the CMD-082 message.

The setting of **sh_script_stop_severity** affects the output of the CMD-082 message. If the setting of **sh_script_stop_severity** causes a CMD-081 message, then it takes precedence over CMD-082.

To determine the current value of this variable, use the **get_app_var sh_source_emits_line_numbers** command.

## SEE ALSO

get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
sh_script_stop_severity(3)
CMD-081(n)
CMD-082(n)

# sh_source_logging

Indicates if individual commands from a sourced script should be logged to the command log file.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

When you source a script, the **source** command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment. You can disable this logging by setting **sh_source_logging** to **false**.

To determine the current value of this variable, use the **get_app_var sh_source_logging** command.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
```

# sh_source_uses_search_path

Indicates if the **source** command uses the **search_path** variable to search for files.

## TYPE

Boolean

## DEFAULT

application specific

## DESCRIPTION

When this variable is set to \ftrue the **source** command uses the **search_path** variable to search for files. When set to **false**, the **source** command considers its file argument literally.

To determine the current value of this variable, use the **get_app_var sh_source_uses_search_path** command.

## SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
search_path(3)
```

# sh_tcllib_app_dirname

Indicates the name of a directory where application-specific Tcl files are found.

## TYPE

string

## DESCRIPTION

The **sh_tcllib_app_dirname** variable is set by the application to indicate the directory where application-specific Tcl files and packages are found. This is a read-only variable.

## SEE ALSO

`get_app_var(2)`

# sh_user_man_path

Indicates a directory root where you can store man pages for display with the **man** command.

## TYPE

list

## DEFAULT

empty list

## DESCRIPTION

The **sh_user_man_path** variable is used to indicate a directory root where you can store man pages for display with the **man** command. The directory structure must start with a directory named *man*. Below *man* are directories named *cat1*, *cat2*, *cat3*, and so on. The **man** command will look in these directories for files named *file.1*, *file.2*, and *file.3*, respectively. These are pre-formatted files. It is up to you to format the files. The **man** command effectively just types the file.

These man pages could be for your Tcl procedures. The combination of defining help for your Tcl procedures with the **define_proc_attributes** command, and keeping a manual page for the same procedures allows you to fully document your application extensions.

The **man** command will look in **sh_user_man_path** after first looking in application-defined paths. The user-defined paths are consulted only if no matches are found in the application-defined paths.

To determine the current value of this variable, use the **get_app_var sh_user_man_path** command.

## SEE ALSO

```
define_proc_attributes(2)
get_app_var(2)
man(2)
set_app_var(2)
```

# synopsys_program_name

Indicates the name of the program currently running.

## TYPE

string

## DESCRIPTION

This variable is read only, and is set by the application to indicate the name of the program you are running. This is useful when writing scripts that are mostly common between some applications, but contain some differences based on the application.

To determine the current value of this variable, use **get_app_var synopsys_program_name**.

## SEE ALSO

get_app_var(2)

# synopsys_root

Indicates the root directory from which the application was run.

## TYPE

string

## DESCRIPTION

This variable is read only, and is set by the application to indicate the root directory from which the application was run.

To determine the current value of this variable, use **get_app_var synopsys_root**.

## SEE ALSO

get_app_var(2)