Library Compiler™
Tool Invocation Commands

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_shell

Runs the Library Compiler command shell.

SYNTAX

lc_shell

[-f *script_file*] [-x *command_string*] [-no_init] [-version]

Data Types

script_file	string
command_string	string

ARGUMENTS

-f *script_file*

Executes *script_file* (a file of **lc_shell** commands) before displaying the initial **lc_shell** prompt. If the last statement in *script_file* is **quit**, no prompt is displayed and the command shell is exited.

-x *command_string*

Executes the **lc_shell** statement in *command_string* before displaying the initial **lc_shell** prompt. Multiple statements can be entered, each statement separated by a semicolon. See the *Multiple Statement Lines and Multiple Line Statements* subsection of this manual page. If the last statement entered is **quit**, no prompt is displayed and the command shell is exited.

-no_init

Tells the **lc_shell** not to execute any **.synopsys_lc.setup** *startup files*. This option is only used when you have a command log or other script file that you want to include in order to reproduce a previous Library Compiler graphical interface or **lc_shell** session. You can include the script file either by using the **-f** option or by issuing the **include** command from within **lc_shell**.

-version

Displays the version number, build date, site identification number, local administrator, and contact

information, and then exits.

DESCRIPTION

Interprets and executes library compiler commands. The **lc_shell** environment consists of user commands and variables that control the creation and manipulation of libraries

The **lc_shell** executes commands until it is terminated by a **quit** or **exit** command. During interactive mode, you can also terminate the **lc_shell** session by typing Control-d.

To cancel (interrupt) the command currently executing in **lc_shell**, type Control-c. The time it takes for a command to process an interrupt (stop what it is doing and continue with the next command) depends upon the size of the library and the type of command. If you enter Control-c three times before a command responds to the interrupt, **lc_shell** exits and the following message is displayed:

Information: Process terminated by interrupt.

There are three basic types of statements in **lc_shell**:

- assignment
- control
- command

Additionally, there are seven types of expressions:

- string
- numeric
- constant
- variable
- list
- command
- operator
- complex

Statements and expressions are discussed in detail in the following subsections.

Special Characters

```
The pipe character ( | ) has no meaning in lc_shell.  Use the
backslash (  )
to escape double quotes when executing a UNIX command.  For example, the
following command requires backslash characters before the double quotes
to prevent Design Compiler from ending the command prematurely:

lc_shell> sh \'grep \'foo\' my_file\'.
```

Control Statements

The two control statements **if** and **while** allow conditional execution and looping in the **lc_shell** language. The syntax of the basic **if** statement is:

```
if ( condition ) {
statement_list
}
```

Other forms of the **if** statement allow use of **else** and **else if**. See the description of the **if** statement in the *Synopsys Commands* section of this manual for details.

The syntax of the **while** statement is:

```
while ( condition ) {
statement_list
}
```

See the description of the **while** statement in the *Synopsys Commands* section of this manual for more details. See the *Operator Expressions* and *Complex Expressions* subsections of this manual page for a discussion of relational and logical operators used in the control statements.

Command Statements

The **lc_shell** invokes the specified command with its arguments. The syntax of a command statement is:

```
command_name argument_1 argument_2 ... argument_n
```

Arguments are separated by commas or spaces and can be enclosed in parentheses. Following are examples of **lc_shell** command statements: lc_shell> **read_lib my_lib.lib** lc_shell> **report_lib my_lib**

String Expressions

A string expression is a sequence of characters enclosed within quotation marks (""). Following are examples of string expressions: "my_lib_name" "~/dir_1/dir_1/file_name" "this is a string"

Numeric Constant Expressions

Numeric constant expressions are numeric values. They must begin with a digit and can contain a decimal point; a leading sign can be included. Exponential notation is also recognized. Following are examples of numeric constant expressions: 123 -234.5 123.4e56

Variable Expressions

A variable expression recalls the value of a previously-defined variable. Variable names can contain letters, digits, and most punctuation characters, but must not start with a digit. Following are examples of variable expressions: current_lib name/name -all +-*/.:'#~`%$&^@!_[]|?

If a variable used in an expression has not previously been assigned a value (in an assignment statement), then its value is a string containing the variable name. This feature allows you to omit the quotes around many strings. For example, the following commands are equivalent (assuming there are no variables called "mylib", "db", \por "-f").

```
lc_shell> write_lib "-f" "db" "mylib"
lc_shell> write_lib -f db mylib
```

List Expressions

A list expression defines a list constant. The list can include pathnames, cell or pin names, values, etc.

The syntax of a list expression is:

```
{ expression_1 expression_2 ... expression_n }
```

Expressions are separated by spaces or commas. Following are examples of list expressions: {} {"pin_1" "pin_2" "pin_3"} {1,2,3,4,5}

Operator Expressions

Operator expressions perform simple arithmetic, and string and list concatenation. The syntax of an operator expression is: expression <operator> expression

where <operator> is: "+", "-", "*", or "/", and is separated by at least one preceding and following space. Operator expressions involving numbers return the computed value. The "+" operator can be used with strings and lists to perform concatenation. Following are examples of operator expressions: 234.23 - 432.1 100 * scale file_name_variable + ".suffix" {portA, portB} + "portC"

The **relational operators** "==", "!=", ">", ">=", "<", and "<=" are used in the control statements **if** and **while**. The "greater than" operator ">" should only be used in parenthesized expressions to avoid confusion with the file redirection operator ">".

The **logical operators** "&&", "||", and "!" (and, or, not) are also used in the control statements **if** and **while**. The "not" operator is different from the other operators in that it is a unary operator with the syntax: ! expression

Complex Expressions

Expressions can be built from other expressions, creating complex expressions. When a complex expression contains more than one operator, **lc_shell** satisfies multiplication and division operators before addition and subtraction. Simple expressions enclosed in parentheses are given priority and override this rule. Thus, the expression "1 + 2 * 3 + 4" has the value 11, and "(1 + 2) * (3 + 4)" has the value 21.

The relational and logical operators can be used in combination to form complex conditions. Following are examples of complex conditional expressions:

```
(goal >= 7.34 || ! complete)
(a >= 7 || run_mode != "test" && !(error_detected == true))
```

Complex logical expressions are evaluated from left to right, with "&&" being evaluated before "||". However, those expressions enclosed in parentheses are evaluated first.

Command Arguments

Many **lc_shell** commands have required or optional arguments that allow you to further define, limit or expand the scope of its operation.

This manual contains a comprehensive list and description of these arguments. You can also use the **man** command to view the manual page online. For example, to view the online manual page of the **read_lib** command, enter: lc_shell> **man read_lib**

Many commands also offer a -help option that lists the arguments available for that command, for example:

```
lc_shell> read_lib -help
Usage: read_lib    # read a .lib file
```

```
[-model_type model file types]
                        (model file type)
[-partial_model_check] (screening for partial model file)
[-signoff_screening]   (screening for sign-off tools)
[-test_model list of CTL files]
                        (list of CTL files)
[-html]                 (report library screener results in html format)
[-no_warnings]          (disable warning messages)
[-lib_messages lib_messages]
                        (predefined lib_message)
[-return_lib_collection]
                        (return library collection)
file_name               (technology library file)
```

Arguments that do not begin with a hyphen (-) are positional arguments. Positional arguments must be entered in a specific order relative to each other. Non-positional arguments (those beginning with a hyphen) can be entered in any order and can be intermingled with positional arguments.

The names of non-positional arguments can be abbreviated to the minimum number of characters required to distinguish them from the other arguments.

The following commands are equivalent: lc_shell> **write_lib -format db -output lib.db my_lib** lc_shell> **write_lib my_lib -format db -output lib.db my_lib** lc_shell> **write_lib -f db -o lib.db my_lib**

Many arguments are optional, but if you omit a required argument, an error message and usage statement are displayed. For example:

```
lc_shell> read_lib
Error: Required argument 'file_name' was not found (CMD-007)
```

Multiple Statement Lines and Multiple Line Statements

Normally, only one command is typed on a single line. If you want to put more than one command on a line, you must separate each command with a semicolon, for example:

```
lc_shell> read_lib my_lib.lib; report_lib my_lib; write_lib my_lib;
```

There is no limit to the number of characters on a **lc_shell** command line, but you can break a long command into multiple lines by terminating all but the last line with a backslash (\e). This tells **lc_shell** to expect the command to continue on the next line:

lc_shell> **read_lib -html \e**
-no_warnings \e
my.lib

This feature is normally used in files containing **lc_shell** commands (*script files*).

Output Redirection

The **lc_shell** lets you divert command output messages to a file. To do this, type "> *file_name*" after any statement. The following example deletes the old contents of "my_file" and writes the output of the **report_lib** command to the file. lc_shell> **report_lib my_lib1 > my_file**

You can append the output of a command to a file with ">>". The following example appends the library report of my_lib2 to the contents of "my_file": lc_shell> **report_lib my_lib2 >> my_file**

Aliases

The **alias** command gives you the ability to define new commands in terms of existing ones. You can reduce the number of keystrokes by defining short aliases for the commands and options you use most often.

The following example defines a new command "q" that is equivalent to running the **quit** command with the -variables option.

```
lc_shell> alias q quit
```

With the "q" alias defined, the following two commands are equivalent:

```
lc_shell> quit
lc_shell> q
```

Alias definitions can be placed in your **.synopsys_lc.setup** file or in a separate file. The advantage of keeping aliases in a separate file is that all defined aliases can be written to a file with a command such as: lc_shell> alias > **~/.synopsys_aliases**

If you put the command **include ~/.synopsys_aliases** in your **.synopsys_lc.setup** file, the aliases are defined every time you start a new **lc_shell** session.

Note that aliases are only expanded if they are the first token in a command. Thus, they can not be used as arguments to other commands. See the description of the **alias** command in the *Synopsys Commands* section of this manual.

History

A record is kept of all **lc_shell** commands issued during any given **lc_shell** session. The **history** command displays a list of these commands. lc_shell> **history** 1 read_lib file.lib 2 report_lib my_lib 3 write_lib my_lib ...

Your previous commands can be re-executed with the following "!" commands:

!!

Expands to the previous command.

!*number*

Expands to the command whose number in the history list matches *number*.

!-*number*

Expands to the command whose number in the history list matches the current command minus *number*.

!*text*

Expands to the most recent command that starts with *text*. A *text* command can contain letters, digits, and underscores, and must begin with a letter or underscore.

!?*text*

Expands to the most recent command that contains *text*. A *text* command can contain letters, digits, and underscores, and must begin with a letter or underscore.

As with aliases, a "!" command must be the first token in a statement, but not necessarily the only one.

```
lc_shell> read_lib file.lib
lc_shell> write_lib my_lib
lc_shell> !! -f db /* Rewrite with the -format db option */
lc_shell> history
      1  read_lib file.lib
      2  write_lib my_lib
      3  write_lib my_lib -f db
      4  history
```

Given the previous history, the following commands are equivalent:

```
lc_shell> !-4 -s file /* Same as command 1 */
lc_shell> !1 -s file
lc_shell> !re -s file
lc_shell> !?lib -s file
lc_shell> !?ead -s file
```

Additional parameters can be included in a **!** command statement. The above examples include the **-single_file** option (**-s file**) of the **read** command.

More than one **!** command can appear in a line, as long as each is the first token in a statement. lc_shell> **!?q; !c; !4**

The previous command is the same as: lc_shell> **read_lib file.lib** lc_shell> **write_lib my_lib -f db** lc_shell> **history**

SEE ALSO

library_compiler(1)
alias(2)
history(2)
if(2)
include(2)
while(2)

synopsys_users

Lists the current users of the Synopsys licensed features.

SYNTAX

synopsys_users [*feature_list*]

list *feature_list*

ARGUMENTS

feature_list

List of licensed features for which to obtain the information. Refer to the Synopsys *System Installation and Configuration Guide* for a list of features supported by the current release. Or, determine from the key file all the features that are licensed at your site.

DESCRIPTION

Displays information about all of the licenses, related users, and hostnames currently in use. If a feature is specified, all users of that feature are displayed.

synopsys_users is valid only when Network Licensing is enabled.

For more information about **synopsys_users**, refer to the *System Installation and Configuration Guide*.

EXAMPLES

In this example, all of the users of the Synopsys features are displayed:

```
% synopsys_users

krig@node1     Design-Analyzer, Design-Compiler, LSI-Interface
                    DFT-Compiler, VHDL-Compiler
doris@node2    HDL-Compiler, Library-Compiler
test@node3     Design-Compiler, Design-Analyzer, TDL-Interface

3 users listed.
```

This example shows users of the "Library-Compiler" or "VHDL-Compiler" feature.

```
% synopsys_users Library-Compiler VHDL-Compiler

krig@node1     Design-Analyzer, Design-Compiler, LSI-Interface
                    DFT-Compiler, VHDL-Compiler
doris@node2    HDL-Compiler, Library-Compiler

2 users listed.
```

SEE ALSO

get_license(2)
license_users(2)
list(2)
remove_license(2)