Milkyway™ Environment Tool 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

Contents

add_to_collection . . . 8
all_connected . . . 11
append_to_collection . . . 13
attach_file . . . 15
begin_scheme . . . 17
change_selection . . . 19
check_library . . . 21
check_mw_lib . . . 25
close_mw_cel . . . 27
close_mw_lib . . . 29
collection_to_list . . . 31
collections . . . 33
compare_collections . . . 38
compute_polygons . . . 40
connect_net . . . 44
convert_fill_polygons . . . 46
convert_from_polygon . . . 48
convert_mw_lib . . . 50
convert_to_polygon . . . 52
convert_wire_ends . . . 54
copy_collection . . . 56
copy_mw_cel . . . 58
copy_mw_lib . . . 61
cputime . . . 63
create_base_array . . . 64
create_boundary . . . 66
create_cell . . . 69
create_mw_bound . . . 72
create_mw_cel . . . 75
create_mw_lib . . . 77
create_net . . . 80
create_net_shape . . . 82
create_placement_blockage . . . 86
create_port . . . 88
create_route_guide . . . 90
create_routing_blockage . . . 93
create_terminal . . . 96
create_text . . . 98
create_track . . . 101

create_user_shape . . . 105
create_via . . . 109
create_via_master . . . 112
create_voltage_area . . . 115
current_instance . . . 118
current_mw_cel . . . 121
current_mw_lib . . . 123
decrypt_lib . . . 125
define_antenna_accumulation_mode . . . 127
define_antenna_area_rule . . . 129
define_antenna_layer_ratio_scale . . . 132
define_antenna_layer_rule . . . 134
define_antenna_rule . . . 137
define_user_attribute . . . 141
detach_file . . . 144
disconnect_net . . . 146
dump_tlu_plus_file . . . 148
enable_ipv6 . . . 150
encrypt_lib . . . 152
extend_mw_layers . . . 154
filter_collection . . . 156
flatten_cell . . . 159
force_exit . . . 162
foreach_in_collection . . . 163
format . . . 165
get_attached_file . . . 169
get_attribute . . . 171
get_bounds . . . 174
get_cells . . . 176
get_layer_attribute . . . 180
get_layers . . . 182
get_mw_cels . . . 184
get_net_shapes . . . 187
get_nets . . . 190
get_physical_lib_cells . . . 193
get_physical_lib_pins . . . 196
get_physical_libs . . . 200
get_pin_guides . . . 203
get_pins . . . 206
get_placement_blockages . . . 210
get_polygon_area . . . 213
get_ports . . . 215
get_preferred_routing_direction . . . 218
get_program_info . . . 220
get_route_guides . . . 222

get_routing_blockages . . . 224
get_selection . . . 228
get_terminals . . . 230
get_texts . . . 233
get_tracks . . . 236
get_user_grid . . . 240
get_user_shapes . . . 242
get_via_masters . . . 247
get_via_regions . . . 249
get_vias . . . 253
get_voltage_areas . . . 255
index_collection . . . 258
list_attributes . . . 260
list_mw_cels . . . 263
load . . . 265
man . . . 266
mem . . . 268
move_objects . . . 269
open_mw_cel . . . 271
open_mw_lib . . . 273
purge_attached_file . . . 275
query_objects . . . 278
read_def . . . 281
read_gds . . . 285
read_icc2_frame . . . 288
read_lef . . . 290
read_oasis . . . 293
read_verilog . . . 296
rebuild_mw_lib . . . 298
remove_antenna_rules . . . 300
remove_attachment_file . . . 302
remove_attribute . . . 306
remove_base_arrays . . . 309
remove_cell . . . 311
remove_collections . . . 313
remove_from_collection . . . 315
remove_lib_cel . . . 317
remove_mw_bounds . . . 319
remove_mw_cel . . . 321
remove_net . . . 324
remove_net_shape . . . 326
remove_object . . . 328
remove_pin_guides . . . 330
remove_placement_blockage . . . 332
remove_port . . . 334

remove_route_guide . . . 336
remove_routing_blockage . . . 338
remove_symbol_table . . . 340
remove_terminal . . . 342
remove_text . . . 344
remove_track . . . 346
remove_user_shape . . . 349
remove_via . . . 351
remove_via_master . . . 353
remove_voltage_area . . . 355
rename_mw_cel . . . 357
rename_mw_lib . . . 359
replace_cell . . . 361
replace_tlu_plus_file . . . 363
report_antenna_rules . . . 365
report_attribute . . . 367
report_check_library_options . . . 370
report_collections . . . 372
report_milkyway_version . . . 374
report_mw_cel . . . 377
report_mw_lib . . . 379
report_track . . . 381
report_voltage_area . . . 384
resize_polygon . . . 387
restore_design_settings . . . 390
rotate_objects . . . 392
save_design_settings . . . 394
save_mw_cel . . . 396
scheme . . . 398
set_attached_file_link_mode . . . 399
set_attribute . . . 401
set_check_library_physical_options . . . 403
set_hierarchy_separator . . . 417
set_mw_lib_reference . . . 419
set_mw_technology_file . . . 421
set_preferred_routing_direction . . . 423
set_user_grid . . . 425
set_via_array_size . . . 428
sizeof_collection . . . 430
sort_collection . . . 432
suppress_message . . . 434
undefine_user_attribute . . . 436
unset_preferred_routing_direction . . . 438
unsuppress_message . . . 440
update_mw_bound . . . 442

update_mw_port_by_db 444
update_voltage_area 447
write_def 450
write_design_settings 455
write_gds 458
write_lef 463
write_mw_lib_files 466
write_oasis 468
write_verilog 472

add_to_collection

Adds objects to a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection add_to_collection
   base_collection
   object_spec
   [-unique]

collection base_collection
list  object_spec
```

ARGUMENTS

base_collection

Specifies the base collection to which objects are to be added. This collection is copied to the result collection, and objects matching *object_spec* are added to the result collection. *base_collection* can be the empty collection (empty string), subject to some constraints, explained in the DESCRIPTION.

object_spec

Specifies a list of named objects or collections to add.

If the base collection is heterogeneous, only collections can be added to it.

If the base collection is homogeneous, the object class of each element in this list must be the same as in the base collection. If it is not the same class, it is ignored. From heterogeneous collections in the *object_spec*, only objects of the same class of the base collection are added. If the name matches an existing collection, the collection is used. Otherwise, the objects are searched for in the database using the object class of the base collection.

The *object_spec* has some special rules when the base collection is empty, as explained in the DESCRIPTION.

-unique

Indicates that duplicate objects are to be removed from the resulting collection. By default, duplicate objects are not removed.

DESCRIPTION

The **add_to_collection** command allows you to add elements to a collection. The result is a new collection representing the objects in the *object_spec* added to the objects in the base collection.

Elements that exist in both the base collection and the *object_spec*, are duplicated in the resulting collection. Duplicates are not removed unless you use the **-unique** option. If the *object_spec* is empty, the result is a copy of the base collection.

If the base collection is homogeneous, the command searches in the database for any elements of the *object_spec* that are not collections, using the object class of the base collection. If the base collection is heterogeneous, all implicit elements of the *object_spec* are ignored.

When the *base_collection* argument is the empty collection, some special rules apply to the *object_spec*. If the *object_spec* is non-empty, there must be at least one homogeneous collection somewhere in the *object_spec* list (its position in the list does not matter). The first homogeneous collection in the *object_spec* list becomes the base collection and sets the object class for the function. The examples show the different errors and warnings that can be generated.

For background on collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example from PrimeTime (using the **get_ports** command) gets all ports beginning with 'mode', then adds the "CLOCK" port.

```
prompt> set xports [get_ports mode*]
{"mode[0]", "mode[1]", "mode[2]"}
prompt> add_to_collection $xports [get_ports CLOCK]
{"mode[0]", "mode[1]", "mode[2]", "CLOCK"}
```

The following example from PrimeTime adds the cell u1 to a collection containing the SCANOUT port.

```
prompt> set sp [get_ports SCANOUT]
{"SCANOUT"}
prompt> set het [get_cells u1]
{"u1"}
prompt> query_objects -verbose [add_to_collection $sp $het]
{"port:SCANOUT", "cell:u1"}
```

The following examples show how **add_to_collection** behaves when the base collection is empty. Adding two empty collections yields the empty collection. Adding an implicit list of only strings or heterogeneous collections to the empty collection generates an error message, because no homogeneous collections are present in the *object_spec* list. Finally, as long as one homogeneous collection is present in the *object_spec* list, the command succeeds, even though a warning message is generated. The example uses the variable settings from the previous example.

```
prompt> sizeof_collection [add_to_collection "" ""]
0

prompt> set A [add_to_collection "" [list a $het c]]
Error: At least one homogeneous collection required for argument 'object_spec'
        to add_to_collection when the 'collection' argument is empty (SEL-014)

prompt> add_to_collection "" [list a $het $sp]]
Warning: Ignored all implicit elements in argument 'object_spec'
 to add_to_collection because the class of the base collection
 could not be determined (SEL-015)
{"SCANOUT", "u1", "SCANOUT"}
```

SEE ALSO

collections(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)

all_connected

Returns the objects connected to a net, port or pin.

SYNTAX

```
list all_connected
   object
```

Data Types

```
   object  list
```

ARGUMENTS

object

A single object list or an object name which specifies the object whose connections are returned. The object must be a net, port, pin. If a net and a pin or a port have the same name, precedence order is net, pin and port when name is provided.

DESCRIPTION

Given a net, return a list of connected ports and pins. Given a pin or port, return the connected net. A collection is returned.

To connect nets to ports or pins, use **connect_net**. To break connections, use **disconnect_net**.

EXAMPLES

The following examples use **all_connected** to list the objects connected to MY_NET:

```
mw_shell> all_connected [get_nets MY_NET]

mw_shell> connect_net MY_NET [get_ports OUT3]
Connecting net 'MY_NET' to port 'OUT3'.

mw_shell> connect_net MY_NET [get_pins U65/Z]
Connecting net 'MY_NET' to pin 'U65/Z'.

mw_shell> all_connected MY_NET
{"OUT3", "U65/Z"}

mw_shell> all_connected OUT3
{"MY_NET"}

mw_shell> all_connected U65/Z
{"MY_NET"}
```

SEE ALSO

connect_net(2)
create_net(2)
current_design(2)
disconnect_net(2)
get_nets(2)
remove_net(2)

append_to_collection

Add object(s) to a collection. Modifies variable.

SYNTAX

```
collection add_to_collection
   var_name
   object_spec
   [-unique]

collection var_name
list    object_spec
```

ARGUMENTS

var_name

Specifies a variable name. The objects matching *object_spec* are added into the collection referenced by this variable.

object_spec

Specifies a list of named objects or collections to add.

-unique

Indicates that duplicate objects are to be removed from the resulting collection. By default, duplicate objects are not removed.

DESCRIPTION

The **append_to_collection** command allows you to add elements to a collection. This command treats the variable name given by *var_name* as a collection, and appends all of the elements in *object_spec* to that collection. If the variable does not exist, it is created as a collection with elements from *object_spec* as its value. If the variable does exist, and it does not contain a collection, it is an error.

The result of the command is the collection which was initially referenced by *var_name*, or the collection created if the variable did not exist.

The **append_to_collection** command is provides the same semantics as a common use of **add_to_collection** but with significant improvement in performance. An example of replacing **add_to_collection** with **append_to_collection** is given below.

```
set var_name [add_to_collection $var_name $objs]
```

Using add_to_collection this becomes:

```
append_to_collection var_name $objs
```

The **append_to_collection** command can be much more efficient than **add_to_collection** if you are building up a collection in a loop. The arguments of the command have the same restrictions as the **add_to_collection** command. Please see the **add_to_collection** man page for more information about those restrictions.

EXAMPLES

The following example from PrimeTime shows how a collection can be built up with **append_to_collection**:

```
pt_shell> set xports
Error: can't read "xports": no such variable
        Use error_info for more info. (CMD-013)
pt_shell> append_to_collection xports [get_ports in*]
{"in0", "in1", "in2"}
pt_shell> append_to_collection xports CLOCK
{"in0", "in1", "in2", "CLOCK"}
```

SEE ALSO

add_to_collection(2)
foreach_in_collection(2)
index_collection(2)
remove_from_collection(2)
sizeof_collection(2)

attach_file

Attaches a file to a library/design.

SYNTAX

```
int attach_file [-class class_name] [-to object_spec] [-comment text] [-pseudonym name] [-copy | -refe

string  class_name
list object_spec
string text
string  name
string file_name
```

ARGUMENTS

-class *class_name*

Specifies the object's class name.

-to *object_spec*

Specifies a library/design object to which the file would be attached. object_spec can be either a collection of exact one object or the name pattern with class specifed by option '-class'. The specifies object should be opened in write(W) mode. If no object is specified, it is determined from the current design firstly, or from the opened main library.

-comment *text*

Text that is to be associated with the attached file. The text string should be within 32-characters.

-pseudonym *name*

Pseudonym to be associated with the attached file. The string should be with 16-characters or less. If the name is not specified, the default name will be given with the string default_N(N is an integer).

-copy

Copies the original file as the attach file of the specified object if the option is set to be true. The default operation is to attach the file on to the specified object and delete the file from its original location.

This argument **-copy** and *-reference* are mutually exclusive.

-reference

Puts a reference to the original file as the attach file of the specified object. The attach file will go to invalid when the original file is removed.

This argument **-copy** and *-reference* are mutually exclusive.

file_name

Specifies the name of the actual file to be attached to. After successfully attached, the actual file will be removed from current location.

DESCRIPTION

The **attach_file** command attaches a file to a specified library/design. If the command is executed successfully, the return value will be set to 1. If some error occurrs, the return value is set to 0.

EXAMPLES

The following example attaches a file to the current design with the default pseudonym.

```
prompt> attach_file attach1
Attaches file default_1 to design 'top'.
1

In this example, the pseudonym is provided by the command.

prompt> exec touch attach2
prompt> attach_file -pseudonym new_attach1 -to [get_libs design] attach2
Attaches file new_attach1 to library 'design'.
1
```

SEE ALSO

detach_file(2)

begin_scheme

Changes the command interpreter to accept Scheme instead of Tcl.

SYNTAX

begin_scheme

DESCRIPTION

This command stops the tool from interpreting typed-in commands as Tcl commands, and causes the tool to begin interpreting them as Scheme commands. Use this command when you have multiple scheme commands to process.

Note that there is no prompt when running in the scheme mode. Many scheme commands return **#t** or **#f** when they complete, corresponding to **1** or **0** from Tcl commands.

If you have only one scheme command to process, use the **scheme** command instead of **begin_scheme**.

When you are finished with entering scheme commands, use the **begin_tcl** scheme command to resume Tcl processing.

EXAMPLES

The following example switches to scheme mode.

```
prompt> begin_scheme
```

SEE ALSO

```
scheme(2)
```

change_selection

Takes a collection of objects and changes the selection in the GUI according to the type of change specified. For use in Tcl-based GUIs only.

SYNTAX

```
status_value change_selection
   [-replace | -add | -remove]
   [collections]

collections list
```

ARGUMENTS

-replace

Replaces the current selection with the objects specified in the collection. This is the default.

-add

Adds the objects specified in the collection to the current selection.

-remove

Removes the objects specified in the collection from current selection.

collections

Specifies the collection of objects to use to change the selection. Currently, the supported types of collections are the following:

```
cell
net
port
pin
terminal
net_shape
route_shape
route_guide
placement_blockage
bound
```

DESCRIPTION

This command changes the selection in the GUI. When selections are changed, the GUI will update all relevant windows to reflect this change.

A collection of objects and the type of change are given as input to the command. This collection might be returned as the result of another command, such as the **get_cells** command. The current selection is cleared is any of the following conditions occur:

- The collection is empty.
- No collection is specified, and the **-replace** option is specified.
- No collection is specified, and the the default is implemented.

For more information about collections, see the **collections** man page.

EXAMPLES

The following example clears the selection.

```
prompt> change_selection
1
```

The following example adds cells to the selection.

```
prompt> change_selection [get_cells *] -add
1
```

SEE ALSO

get_selection(2)

check_library

Checks and reports the data and the quality for the specified library after running the set_check_library_physical_options command.

SYNTAX

```
int check_library
    [ -mw_lib_name phys_lib_name ]
    [ -output file_name ]
```

Data Types

phys_lib_name string
file_name string

ARGUMENTS

-mw_lib_name *phys_lib_name*

Specifies the Milkyway library to check. When checking tech consistency and same name cells, use the main or design library name. The checks are performed on the main library and its reference libraries. When specifying other types of checks, the checks are performed on only the specified library. If the library is not specified, the current library is used.

-output *file_name*

Specifies the name of the file in which to save the check report. This file is different from the default Milkyway log file. If a file name is not specified, the report is only displayed.

DESCRIPTION

The **check_library -mw_lib_name** command checks and reports the following in the specified library:
(1) Technology data consistency between the specified main library and each linked

```
      reference library.
   (2) Cell view vs. FRAM view in the library for missing views and mismatched views.
   (3) Cells with identical names in different reference libraries and the specified
       main library.
   (4) Signal EM rule
   (5) Antenna rules and missing antenna properties
   (6) Rectilinear cells
   (7) Physical only cells
   (8) Physical properties for P & R including unit tiles for the libraries
   (9) Pin accessibility/routeability
   (10) Technology data quality
   (11) DRC for each cell in the library (FRAM view)
   (12) Macro metal density for specified cells
   For details about using set_check_library_physical_options to set the types of
   checking to be performed, see the man page for that command.
   The report header shows the library name and the time that the library checking
   is performed.
   Check the entire library after the library prepration is finished.
   Before issuing check_library, you must use set_check_library_physical_options
   to set the options.
   A status indicating success or failure is returned.
```

EXAMPLES

The following example checks all data for the specified library test.mw.

```
prompt> set_check_library_physical_options -all
1
prompt> check_library -mw_lib_name test.mw
1

#BEGIN_CHECK_LIBRARY

   Main library name: /remote/reg_designs/DATA_CENTER/Feature/XIG/library/test.mw
   Check date and time: Tue Jan 23 11:34:34 2007

#BEGIN_CHECK_TECH_CONSISTENCY

   No technology inconsistency found.

#END_CHECK_TECH_CONSISTENCY

#BEGIN_CHECK_TECH

Warning: Layer 'METAL1' has a pitch 0.41 that does not match the recommended
  wire-to-via pitch 0.405. (TFCHK-049)
Warning: ContactCode 'CONT1' is missing the attribute 'unitMinResistance'.
  (line 5559) (TFCHK-014)

#END_CHECK_TECH

#BEGIN_CHECK_VIEWCMP

   Total number of cells missing CEL view:  0 (out of 997)
   Total number of cells missing FRAM view: 1 (out of 997)
   Total number of cells with mismatched view (CEL vs. FRAM):   0 (out of 997)

   X - cell missing view in the Table
```

```
      List of cells with missing views
  ------------------------------------------------------
  Cell Name                CEL              FRAM
  ------------------------------------------------------
  cell_1                   cell_1:1          X
  ------------------------------------------------------

#END_CHECK_VIEWCMP

#BEGIN_CHECK_SAMENAMECELL

   Total number of cells with same names:   2 (out of 1193)

      List of cells with same names
  ------------------------------------------------------------
   Cell Name                 Library list
  ------------------------------------------------------------
   XOR3                      mainlib ref ref1 ref3 ref5
   DFF                       ref ref1 ref3
  ------------------------------------------------------------

#END_CHECK_SAMENAMECELL

#BEGIN_CHECK_SIGNALEM

   The library is missing signal EM data.

#END_CHECK_SIGNALEM

#BEGIN_CHECK_ANTENNA

   The library is missing antenna rules.
   Total number of cells missing hierarchical antenna properties:       1 (out of 53)

              List of cells missing antenna property
  ------------------------------------------------------------------------------
   Cell name             Cell type  Missing property     Pin name(s)
  ------------------------------------------------------------------------------
   ram256x27             Macro      HierAntenna          Q[16] Q[17] Q[18] Q[19]
  ------------------------------------------------------------------------------

#END_CHECK_ANTENNA

#BEGIN_CHECK_RECTILINEARCELL

   Total number of rectilinear cells:   0 (out of 53)

#END_CHECK_RECTILINEARCELL

#BEGIN_CHECK_PHYSICALONLYCELL

   Total number of physical only cells: 0 (out of 53)

#END_CHECK_PHYSICALONLYCELL

#BEGIN_CHECK_PHYSICALPROPERTY

                List of main and reference libraries
  ------------------------------------------------------------------------------------
  Library name       Path                                        Unit tile  Tile size
  ------------------------------------------------------------------------------------
  test.mw            /remote/reg_designs/DATA_CENTER/Feature/ unit       1.800x14.400
  reflib1            /remote/reg_designs/DATA_CENTER/Feature/ unit       1.800x14.400
```

```
  ------------------------------------------------------------------------------------

                List of tile patterns
  ------------------------------------------------------------------------------------
  Tile pattern       Unit tile       Location       Orientation          Tile size
  ------------------------------------------------------------------------------------
  1                  unit            (0,0)          R0|R0_MX             1.800x14.400
  ------------------------------------------------------------------------------------

       List of placement properties
  ------------------------------------------------------------------------------------
   Cell name   PR boundary      Height  Symmetry  Orientation   Tile pattern  Remarks
  ------------------------------------------------------------------------------------
   DFFX1       (0,0)(11.2,5.04)  1xH    x         R0|R0_MY|R180      1
  ------------------------------------------------------------------------------------

       List of routing properties
  ------------------------------------------------------------------
              Preferred      Track
   Layer      direction    direction   Offset    Pitch   Remarks
  ------------------------------------------------------------------
   METAL1         H            H       0.280     0.560   OK
   METAL2         V            V       0.330     0.660   OK
  ------------------------------------------------------------------

#END_CHECK_PHYSICALPROPERTY

#BEGIN_CHECK_ROUTEABILITY

   Total number of pins without on-track routeability:  0 (out of 53)

#END_CHECK_ROUTEABILITY

#BEGIN_CHECK_DRC

   Total number of cells with DRC violations: 0 (out of 53)

#END_CHECK_DRC

#END_CHECK_LIBRARY

1
```

SEE ALSO

set_check_library_physical_options(2)
report_check_library_options(2)
check_mw_lib(2)

check_mw_lib

Check Milkyway library.

SYNTAX

```
status_value check_mw_lib
   [mw_lib | -lib_id lib_id]
   [-technology]
   [-design_drc [-treat_blockage_as_thin_wire]]
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be checked. This argument and **-lib_id** are mutually exclusive. If both are omitted, the current Milkyway library is used.

-lib_id *lib_id*

Specifies the ID of the Milkyway library to be checked. This argument and *mw_lib* are mutually exclusive. If both are omitted, the current Milkyway library is used.

-design_drc

Indicates to check that .FRAM cell data is complete and valid.

-technology

Indicates to check that the technology file is loaded and valid.

-treat_blockage_as_thin_wire

Specifies to treat all blockages as thin wire instead of classifying them according to their widths.

DESCRIPTION

Check various information about a Milkyway library.

A status indicating success or failure is returned.

EXAMPLES

The following example checks technology of the current Milkyway library.

```
prompt> check_mw_lib -technology
1
```

The following example checks design drc of the current Milkyway library.

```
prompt> check_mw_lib -design_drc
---------- Begin of Cell DRC ----------


Total of 0 cells scanned, including
                    0 Macro cells
                    0 Module cells
                    0 IOPad cells
                    0 Standard cells
                    0 Cover cells
                    0 DoubleHeightStd cells
                    0 CornerPad cells

Total of 0 scanned cells failed DRC.

Elapsed =    0:00:00, CPU =    0:00:00

---------- End of Cell DRC ----------

1
```

SEE ALSO

close_mw_lib(2)
open_mw_lib(2)

close_mw_cel

Closes the specified Milkyway cels.

SYNTAX

```
status_value close_mw_cel
   [-save]
   mw_cel_list
```

Data Types

```
   mw_cel_list   list
```

ARGUMENTS

-save

Indicates that the specified Milkyway cels are to be saved before closing. By default, this command discards any changes made to the cel and closes the specified Milkyway cel.

mw_cel_list

Specifies the Milkyway cels to be removed. You can specify Milkyway cels by name, name pattern, or by the Milkyway cels collection's name. For example, specifying *top* matches an Milkyway cels named *top* in the current library. Specifying *top** matches all Milkyway cels having names beginning with *top*. The command **close_mw_cel [get_mw_cels *]** closes all Milkyway cels in the current library.

If no Milkyway cel is specified, the current Milkyway cel is used.

DESCRIPTION

This command saves or discards the specified Milkyway cels or the current Milkyway cel, and then closes them.

If the closed Milkyway cel is the current Milkyway cel, this command clears the Milkyway cel automatically.

EXAMPLES

The following example discards and closes the current Milkyway cel.

```
prompt> close_mw_cel
1
```

The following example saves and closes a list of Milkyway cels.

```
prompt> close_mw_cel -save {test1 test2}
Info: Saved "test1.CEL;1".
Info: Saved "test2.CEL;1".
1
```

The following example discards and closes a list of Milkyway cels.

```
prompt> close_mw_cel {test1 test2}
1
```

SEE ALSO

copy_mw_cel(2)
create_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
remove_mw_cel(2)
rename_mw_cel(2)

close_mw_lib

Closes the current Milkyway library.

SYNTAX

```
status close_mw_lib
   [-save]
```

ARGUMENTS

-save

Saves all Milkyway designs opened in this library. By default, the command discards changes of Milkyway designs in this library.

DESCRIPTION

This command closes the current Milkyway library. It returns a status indicating success or failure.

EXAMPLES

The following example closes the current Milkyway library.

```
prompt> close_mw_lib
1
```

SEE ALSO

open_mw_lib(2)

collection_to_list

Converts a collection to a Tcl list.

SYNTAX

```
list collection_to_list
   [-quiet]
   [-no_duplicates]
   [-base_name]
   [-name_only]
   [-classes class_name]
   collection
```

ARGUMENTS

-quiet

Suppresses the messages warning about unmatched objects or object patterns.

-no_duplicates

Removes duplicate objects from the output list.

-base_name

Shows the base name of the objects (as stored in the **base_name** attribute), instead of the full path name (as stored in the **full_name** attribute). By default, the full path name is shown.

-name_only

Shows only the object name, without its class. By default, this command creates a list for each object. The list has the following format:

```
{ class_name object_name }
```

-classes *class_names*

Show only objects whose class matches one of the specified *class_names*. The valid *class_names* values are **_instance**, **_reference**, **_net**, **_net_shape**, **_pin**, **_port**, **_port_shape** or **_ct_domain**. Use a Tcl list to specify multiple classes. If you do not include this option, the command shows all classes.

collection

Specifies the collections to convert to a Tcl list.

DESCRIPTION

The **collection_to_list** command converts a collection to a Tcl list, which can then be assigned to variables and used by other commands or processed in Tcl scripts.

EXAMPLES

The following examples convert collections to lists.

```
prompt> current_instance VCT
VCT

prompt> collection_to_list [get_instances *]
{ _instance VCT/U1 } { _instance VCT/X1 } { _instance VCT/Y1 }

prompt> collection_to_list -name_only [get_instances U*]
VCT/U1

prompt> collection_to_list -base_name "[get_instances U*] \
? [get_instances X*]"
{ _instance U1 } { _instance X1 }

prompt> collection_to_list -base_name -name_only [get_ports *]
P10 P11 P12

prompt> collection_to_list -base_name -class _port \
? " [get_instances *] [get_ports *]"
{ _port P10 } { _port P11 } { _port P12 }
```

SEE ALSO

collection(2)
get_ct_domains(2)
get_instances(2)
get_net_shapes(2)
get_nets(2)
get_pins(2)
get_port_shapes(2)
get_ports(2)
get_references(2)

collections

Describes the methodology for creating collections of objects and querying objects in the database.

DESCRIPTION

Synopsys applications build an internal database of the netlist and the attributes applied to it. This database consists of several classes of objects; including designs, libraries, ports, cells, nets, pins, and physical classes of objects such as net_shapes, route_shapes, placement_blockages, route_guides, terminals and bounds. Most commands operate on these objects.

By definition:

A collection is a group of objects exported to the Tcl user interface.

Collections have an internal representation (the objects) and, sometimes, a string representation. The string representation is generally used only for error messages.

A set of commands to create and manipulate collections is provided as an integral part of the user interface. The collection commands encompass two categories: those that create collections of objects for use by another command, and one that queries objects for viewing. The result of a command that creates a collection is a Tcl object that can be passed along to another command. For a query command, although the visible output looks like a list of objects (a list of object names is displayed), the result is a string.

An empty string "" is equivalent to the empty collection, that is, a collection with zero elements.

Homogeneous and Heterogeneous Collections

A homogeneous collection contains only one type of object. A heterogeneous collection can contain more than one type of object. Commands that accept collections as arguments can accept either type of collection.

Lifetime of a Collection

Collections are active only as long as they are referenced. Typically, a collection is referenced when a variable is set to the result of a command that creates it or when it is passed as an argument to a command or a procedure. For example, if in Astro or Milkyway you save a collection of ports:

```
prompt> set ports [get_ports *]
```

then either of the following two commands deletes the collection referenced by the *ports* variable:

```
prompt> unset ports
prompt> set ports "value"
```

Collections can be implicitly deleted when they go out of scope. Collections go out of scope when the

parent (or other antecedent) of the objects within the collection is out of scope. For example, if our collection of ports is owned by a design, it is implicitly deleted when the design that owns the ports is closed. When a collection is implicitly deleted, the variable that referenced the collection still holds a string representation of the collection. However, this value is useless because the collection is gone, as illustrated in the following example:

```
prompt> current_design
{"TOP"}

prompt> set ports [get_ports in*]
{"in0", "in1"}

prompt> close_design TOP
1

prompt> query_objects $ports
Error: No such collection '_sel26' (SEL-001)
```

Iteration

To iterate over the objects in a collection, use the **foreach_in_collection** command. You cannot use the Tcl-supplied **foreach** iterator to iterate over the objects in a collection, because **foreach** requires a list; and a collection is not a list. In fact, if you use **foreach** on a collection, it will destroy the collection.

The arguments to **foreach_in_collection** are similar to those of foreach: an iterator variable, the collections over which to iterate, and the script to apply at each iteration. Note that unlike **foreach**, the **foreach_in_collection** command does not accept a list of iterator variables.

The following example is an iterative way to perform a query. For details, see the **foreach_in_collection** man page or the user guide.

```
prompt> \
foreach_in_collection s1 $collection {
  echo [get_attr $s1 name]
  }
```

Manipulating Collections

A variety of commands are provided to manipulate collections.

- **add_to_collection** - Takes a base collection and a list of element names or collections that you want to add to it. The base collection can be the empty collection. The result is a new collection. In addition, the **add_to_collection** command allows you to remove duplicate objects from the collection by using the **-unique** option.
- **remove_from_collection** - Takes a base collection and a list of element names or collections that you want to remove from it. For example, in Astro and Milkyway:

  ```
  prompt> set dports \
  [remove_from_collection [get_port *] CLK]
  {"in1", "in2", "in3"}
  ```

- **compare_collections** - Verifies that two collections contain the same objects (optionally, in the same order). The result is "0" on success.
- **copy_collection** - Creates a new collection containing the same objects (in the same order) as a given collection. Not all collections can be copied.

- **index_collection** - Extracts a single object from a collection and creates a new collection containing that object. Not all collections can be indexed.
- **sizeof_collection** - Returns the number of objects in a collection.

Filtering

You can filter any collection by using the **filter_collection** command. It takes a base collection and creates a new collection that includes only those objects that match an expression.

Many of the commands that create collections support a **-filter** option, which allows objects to be filtered out before they are ever included in the collection. Frequently this is more efficient than filtering after the they are included in the collection. The following example filters out all leaf cells:

```
prompt> filter_collection \
[get_cells *] "is_hierarchical == true"]
{"i1", "i2"}
```

The basic form of a filter expression is a series of relations joined together with AND and OR operators. Parentheses are also supported. The basic relation contrasts an attribute name with a value through a relational operator. In the previous example, *is_hierarchical* is the attribute, == is the relational operator, and *true* is the value.

The relational operators are

```
==   Equal
!=   Not equal
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
=~   Matches pattern
!~   Does not match pattern
```

The basic relational rules are

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only true or false.

Additionally, existence relations determine if an attribute is defined or not defined, for the object. For example,

```
sense == setup_clk_rise and defined(sdf_cond)
```

The existence operators are

```
defined
undefined
```

These operators apply to any attribute as long as it is valid for the object class. See the appropriate man pages for complete details.

Sorting Collections

You can sort a collection by using the **sort_collection** command. It takes a base collection and a list of attributes as sort keys. The result is a copy of the base collection sorted by the given keys. Sorting is

ascending, by default, or descending when you specify the **-descending** option. In the following example, the Astro or Milkyway command sorts the ports by direction and then by full name.

```
prompt> sort_collection [get_ports *] \
{direction full_name}
{"in1", "in2", "out1", "out2"}
```

Implicit Query of Collections

All commands that create implicitly collections query the collection when the command is used at the command line. The number of objects displayed is controlled by the **collection_result_display_limit** variable. Consider the following examples from Astro and Milkyway:

```
prompt> remove_port [get_ports in*]
1
prompt> get_ports in*
{"in0", "in1", "in2"}
prompt> query_objects -verbose [get_ports in*]
{"port:in0", "port:in1", "port:in2"}
prompt> set iports [get_ports in*]
{"in0", "in1", "in2"}
```

In the first example, the **get_ports** command creates a collection of ports that is passed to the **remove_port** command. This collection is not the result of the primary command (**remove_port**), so it is not queried. The second example shows how a command that creates a collection automatically queries the collection when that command is used as a primary command. The third example shows the verbose feature of the **query_objects** command, which is not available with implicit query. Finally, the fourth example sets the variable iports to the result of the **get_ports** command. Only in the final example does the collection persist to future commands until iports is overwritten, unset, or goes out of scope.

Controlling Deletion Effort

When a subset of objects in a design is removed, it is not always clear whether to remove the collection. The Astro and Milkyway **collection_deletion_effort** variable controls the amount of effort expended to preserve collections. For complete details, see the **collection_deletion_effort** command man page.

Related Commands

For your convenience, related commands and variables are listed below by categories:

Common Commands:

```
add_to_collection
compare_collections
copy_collection
foreach_in_collection
index_collection
query_objects
remove_from_collection
sizeof_collection
filter_collection
sort_collection
```

Astro/Milkyway Commands

```
all_clocks
all_connected
```

```
all_inputs
all_outputs
get_cells
get_clocks
get_designs
get_lib_cells
get_lib_pins
get_libs
get_nets
get_pins
get_ports

and for physical objects:

get_placement_blockages
get_route_guides
get_route_shapes
get_terminals
```

Astro and Milkyway Variables:

```
collection_deletion_effort
collection_result_display_limit
```

SEE ALSO

```
add_to_collection(2)
all_clocks(2)
all_connected(2)
all_inputs(2)
all_outputs(2)
compare_collections(2)
copy_collection(2)
filter_collection(2)
foreach_in_collection(2)
get_cells(2)
get_clocks(2)
get_designs(2)
get_lib_cells(2)
get_lib_pins(2)
get_libs(2)
get_nets(2)
get_pins(2)
get_ports(2)
index_collection(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
sort_collection(2)
collection_deletion_effort(3)
```

compare_collections

Compares the contents of two collections. If the same objects are in both collections, the result is 0 (like string compare). If they are different, the result is nonzero. The order of the objects can optionally be considered.

SYNTAX

```
int compare_collections
   [-order_dependent]
   collection1
   collection2


collection1  collection
collection2  collection
```

ARGUMENTS

-order_dependent

Indicates that the order of the objects is to be considered. The collections are considered to be different if the objects are ordered differently.

collection1

Specifies the base collection for the comparison. The empty string (the empty collection) is a legal value for the *collection1* argument.

collection2

Specifies the collection with which to compare to *collection1*. The empty string (the empty collection) is a legal value for the *collection2* argument.

DESCRIPTION

The **compare_collections** command compares the contents of two collections. By default, the order of

the objects does not matter, so that a collection of cells u1 and u2 is the same as a collection of the cells u2 and u1. By using the **-order_dependent** option, the order of the objects is considered.

Either or both of the collections can be the empty string (the empty collection). If two empty collections are compared, the comparison succeeds (**compare_collections** considers them identical), and the result is 0.

EXAMPLES

The following example shows a variety of comparisons. A result of 0 from **compare_collections** indicates success. Any other result indicates failure.

```
prompt> compare_collections [get_cells *] [get_cells *]
0

prompt> set c1 [get_cells {u1 u2}]
{"u1", "u2"}

prompt> set c2 [get_cells {u2 u1}]
{"u2", "u1"}

prompt> set c3 [get_cells {u2 u4 u6}]
{"u2", "u4", "u6"}

prompt> compare_collections $c1 $c2
0

prompt> compare_collections $c1 $c2 -order_dependent
-1

prompt> compare_collections $c1 $c3
-1
```

The following example builds on the previous example by showing how empty collections are compared.

```
prompt> set c4 ""

prompt> compare_collections $c1 $c4
-1

prompt> compare_collections $c4 $c4
0
```

SEE ALSO

collections(2)

compute_polygons

Returns a list of polygons that cover the areas according to the Boolean operation type.

SYNTAX

```
list compute_polygons
   -boolean type
   polygon1
   polygon2
```

Data Types

```
   type           string
   polygon1       list
   polygon2       list
```

ARGUMENTS

-boolean *type*

Specify the type of Boolean operation. Type can be *and*, *or*, *not*, *xor*.

If the operation type is *and*, the returned polygons will cover the areas common to the input polygons;

If the operation type is *or*, the returned polygons will cover the areas, which are covered by either polygon or both polygons;

If the operation type is *not*, the returned polygons will cover the areas, which are covered by the first polygon, but not covered by the second polygon;

If the operation type is *xor*, the returned polygons will cover the areas, which are covered by either polygon, but not both polygons.

polygon1

Specifieds the first polygon represented as a list of points. The format for a polygon is: {{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}. Besides, a list of one polygon is also supported as input for this option, with the format: {{{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}}. Note that the valid polygon is rectilinear, so the adjacent points have one same coordinate. The coordinates' unit is specified in technology file (usually it is micron).

polygon2

Specifieds the second polygon represented as a list of points. The format for a polygon is: {{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}. Besides, a list of one polygon is also supported as input for this option, with the format: {{{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}}. Note that the valid polygon is rectilinear, so the adjacent points have one same coordinate. The coordinates' unit is specified in technology file (usually it is micron).

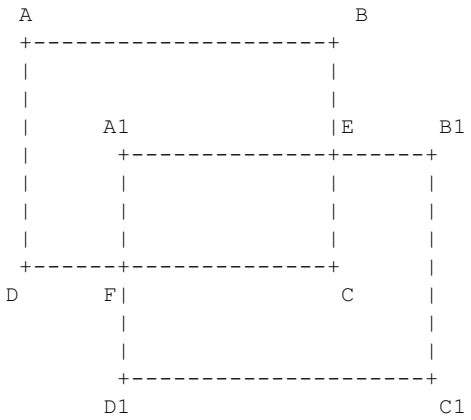
DESCRIPTION

This command gets two outlined polygons and then applies the specified Boolean operation on them. Each outlined polygon is represented as a list of points. The Boolean operation include *or*, *and*, *not*, *xor*.

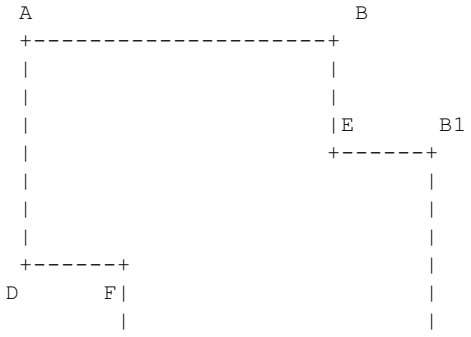
Before this command is used, the library should be opened.

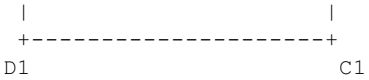
After the operation, the command returns a list of outlined polygons, and the list may consist of one polygon or multiple polygons depending on the result of Boolean operation. Note that for each Boolean operation type, the result could be multiple polygons or one polygon. So do not directly pass the result of this command as a parameter to another polygon command. Tcl list command like **foreach**, **lindex** can be used to extract each polygon from the returned list, and then pass each polygon to other polygon command.

For example: You have two outlined polygons A-B-C-D-A and A1-B1-C1-D1-A1.

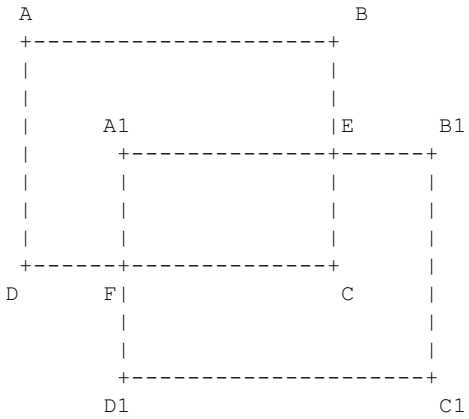


If you use compute_polygons to get the *OR* area of the two outlined polygons, you will get a list of 1 outlined polygon A-B-E-B1-C1-D1-F-D-A.



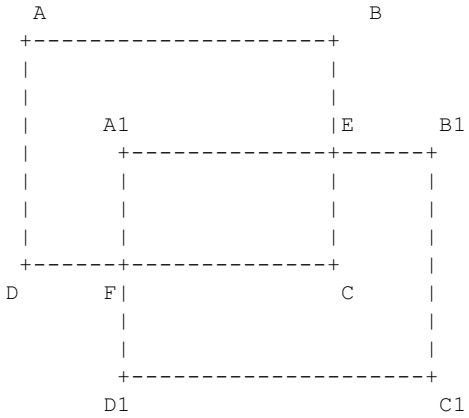


But if you use compute_polygons to get the *XOR* area of the outlined polygons, the command will return two polygons A-B-E-A1-F-D-A and E-B1-C1-D1-F-C-E which cover the areas covered by either polygon A-B-C-D-A or A1-B1-C1-D1-A1, but not by both. So the returned list will consist of two outlined polygons.



EXAMPLES

Using the following two polygons as input example. Polygon A-B-C-D-A: {{0 10} {30 10} {30 30} {0 30} {0 10}}. Polygon A1-B1-C1-D1-A1: {{10 0} {40 0} {40 20} {10 20} {10 0}}.



The following example returns the polygon A1-E-C-F-A1 which covers the common areas of the two input polygons A-B-C-D-A and A1-B1-C1-D1-A1.

```
prompt> compute_polygons -boolean and {{0 10} {30 10} {30 30} {0 30}
{0 10}} {{10 0} {40 0} {40 20} {10 20} {10 0}}
{{10 20} {30 20} {30 10} {10 10} {10 20}}
```

The following example returns the polygon A-B-E-B1-C1-D1-F-D-A which covers the areas covered by either polygon A-B-C-D-A or A1-B1-C1-D1-A1.

```
prompt> compute_polygons -boolean or {{0 10} {30 10} {30 30} {0 30}
{0 10}} {{10 0} {40 0} {40 20} {10 20} {10 0}}
```

```
{{30 30} {30 20} {40 20} {40 0} {10 0} {10 10} {0 10} {0 30} {30 30}}
```

The following example returns the polygon A-B-E-A1-F-D-A which covers the areas covered by the first polygon A-B-C-D-A, but not covered by the second polygon A1-B1-C1-D1-A1.

```
prompt> compute_polygons -boolean not {{0 10} {30 10} {30 30} {0 30}
{0 10}} {{10 0} {40 0} {40 20} {10 20} {10 0}}
{{30 30} {30 20} {10 20} {10 10} {0 10} {0 30} {30 30}}
```

The following example returns two polygons A-B-E-A1-F-D-A and E-B1-C1-D1-F-C-E which cover the areas covered by either polygon A-B-C-D-A or A1-B1-C1-D1-A1, but not by both.

```
prompt> compute_polygons -boolean xor {{0 10} {30 10} {30 30} {0 30}
{0 10}} {{10 0} {40 0} {40 20} {10 20} {10 0}}
{{40 20} {40 0} {10 0} {10 10} {30 10} {30 20} {40 20}} {{30 30}
{30 20} {10 20} {10 10} {0 10} {0 30} {30 30}}
```

The following script example shows how to use the output of the **compute_polygons** command as input for other UI commands.

```
set poly1 [convert_to_polygon [get_net_shapes RECTANGLE#123]]
Set poly2 [convert_to_polygon [get_net_shapes RECTANGLE#456]]
set my_polys [compute_polygons -boolean xor $poly1 $poly2]

foreach poly $my_polys {
    set my_xor_rect [convert_from_polygon $poly]
    echo $my_xor_rect
}
```

SEE ALSO

convert_to_polygon(2)
convert_from_polygon(2)
resize_polygon(2)

connect_net

Connects a specified net to specified pins or ports.

SYNTAX

```
int connect_net
   net
   object_list

net list


object_list list
```

ARGUMENTS

net

Specifies the net to connect. It can be a name or collection. The net must be a scalar (single-bit) net and must exist in the current design.

object_list

Specifies a list of pins and ports to which the net is to be connected. Pins and ports must exist in the current design. If a specified pin or port is already connected, the command issues an error message.

DESCRIPTION

This command connects a net to specified pins or ports. A net can be connected to many pins or ports; however, you cannot connect a pin or port to more than one net.

To disconnect objects on a net, use the **disconnect_net** command. To display pins and ports on a net, use the **all_connected** command.

EXAMPLES

The following example connects a net named *NET0*. The **all_connected** command returns the objects connected to *NET0*.

```
prompt> connect_net NET0 [get_ports A1, A2]


prompt> connect_net NET0 [get_pins U1/A]


prompt> all_connected NET0
{"A1", "A2", "U1/A"}
```

The following example shows the error message generated if you attempt to connect a pin or port to more than one net.

```
prompt> connect_net MY_NET_1 [get_ports PORT1]
Connecting net 'MY_NET_1' to port 'PORT1'.


prompt> connect_net MY_NET_2 [get_ports PORT1]
Error: Object 'PORT1' is already connected to net 'MY_NET_1'.
```

SEE ALSO

```
all_connected(2)
create_net(2)
current_design(2)
disconnect_net(2)
get_nets(2)
remove_net(2)
```

convert_fill_polygons

Converts the input cell's FILL view polygons to rectangles.

SYNTAX

```
status_value convert_fill_polygons
   -library library_name
   -from cell_name
   -to cell_name
```

Data Types

```
   library_name     string
   cell_name        string
```

ARGUMENTS

-library *library_name*

Specifies the library that contains the input and output cell.

-from *cell_name*

Specifies the input cell name.

-to *cell_name*

Specifies the output cell name.

DESCRIPTION

This command converts the input cell's FILL view polygons to rectangles, and then stores the cell into the FILL view of the specified output cell name. The input cell is not actually modified, unless the output cell name is the same as the input cell name.

The command returns a status indicating success or failure.

EXAMPLES

The following example converts the "in_top" cell's FILL view polygons to rectangles and stores the modified cell in the "out_top" cell's FILL view.

```
prompt > convert_fill_polygons -library mylib -from in_top -to out_top
```

The following example converts the "top" cell's FILL view polygons to rectangles. The input cell is the same as the output cell.

```
prompt > convert_fill_polygons -library mylib -from top -to top
```

convert_from_polygon

Fracture a polygon into a list of rectangles, which are mutually exclusive.

SYNTAX

```
list convert_from_polygon
   polygon
   [-format format]
```

Data Types

```
   polygon       list
```

ARGUMENTS

polygon

One polygon which is represented as a list of points. The format for a polygon is: {{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}. Besides, a list of one polygon is also supported as input for this option, with the format: {{{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}}.Pay attention that the valid polygon is rectilinear, so the adjacent points have one same coordinate. The coordinate unit is specified in technology file (usually it is micron).

-format *format*

Specify the output format of result rectangles, *format* can be *polygon* or *rectangle*. If *-format polygon* is specified, each result rectangle will be represented in polygon format like:{{x1 y1} {x2 y2} {x3 y3} {x4 y4} {x1 y1}}; otherwise in default format: {x1 y1} {x2 y2}.

DESCRIPTION

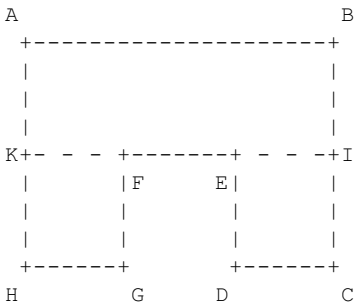
This command converts a polygon into a list of rectangles, which are mutually exclusive. Each returned rectangle will be represented as Tcl list of coordinates.

Before this command is used, the library should be opened. It is important to note that this command may

return a list of rectangles which represent a disjoined rectilinear region. So do not directly pass the result of this command as a parameter to another polygon command. Tcl list command like **foreach**, **lindex** can be used to extract each rectangle from the returned list, and then pass each rectangle to other polygon command.

EXAMPLES

The following example converts the polygon A-B-C-D-E-F-G-H-A to three rectangles K-F-G-H, E-I-C-D and A-B-I-K.




```
Prompt> convert_from_polygon {{0 20} {30 20} {30 0} {20 0} {20 10}
{10 10} {10 0} {0 0} {0 20}}
{{0 10} {10 10} {10 0} {0 0} {0 10}} {{20 10} {30 10} {30 0} {20 0} {20 10}}
{{0 20} {30 20} {30 10} {0 10} {0 20}}
```

SEE ALSO

convert_to_polygon(2)
compute_polygons(2)
resize_polygon(2)

convert_mw_lib

Converts a Milkyway library's cell data.

SYNTAX

```
status convert_mw_lib
   mw_lib
   [ -cell_name <cell_name> ]
   [ -all ]
   [ -check_only ]
```

Data Types

```
   mw_lib string
   cell_name  string
```

ARGUMENTS

mw_lib

Specifies the library containing the cells to be converted.

-cell_name *cell_name*

Specifies the name of cell to be converted. If the option is specified, all child soft macro cells referred by the current cell will be converted along with the current cell.

-all

If the option is specified, all cells under the design lib will be converted.

-check_only

This option lists the names of the cells that need to be converted, together with their library names, without actually performing the conversion. This option can be used only with the -cell_name or -all option. The list generated by the -check_only option includes cells that belong to reference libraries. To convert such cells, you need to apply the convert_mw_lib command directly to the library containing the cells, because the convert_mw_lib command only converts cells contained in the specified library, not cells in reference libraries.

DESCRIPTION

This command converts cells of the specified library. It does not convert cells in the reference libraries.

If a cell was created by a prior release, it is old and must be updated before it is opened by the current release. Once updated, the cells may be used by any tool from the current release.

It is recommended to convert all cells once using this convert_mw_lib command. This minimizes run-time and memory consumption for conversion and would have less impact to the ICC flow. If an old cell is not converted, the open_mw_cel command will open the old cell and automatically convert it in memory. If the user does not save the cell, the cell will be automatically converted again during the next open_mw_cel command call.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

```
convert_mw_lib foo_library -cell_name top
1
```

SEE ALSO

```
open_mw_cel(2)
save_mw_cel(2)
```

convert_to_polygon

Returns a polygon from any supported object.

SYNTAX

```
list convert_to_polygon
   [-quiet]
   object_spec
```

Data Types

```
   object_spec       string
```

ARGUMENTS

-quiet

Indicates that error and warning messages are not to be reported.

object_spec

Specifies a single object from which to get the polygon. This option should be a collection consisting of exactly one supported object.

DESCRIPTION

This command returns a polygon calculated from the input object. The option *object_spec* should be a collection consisting of exactly one object. The supported object classes are *shape*, *placement_blockage*, *route_guide*, and the supported class type can be extended in the future.

The returned polygon is represented as a Tcl list of coordinates. If the input object is polygon type, this command will just output the points of the polygon; if the input object is rectangle or path type, this command will use its information to calculate the corresponding polygon.

EXAMPLES

The following example returns a polygon from net_shape RECTANGLE#123.

```
Promt> convert_to_polygon [get_net_shapes RECTANGLE#123]
{{99.490 111.340} {117.590 111.340} {117.590 111.620} {117.590 99.490}
{99.490 111.340}}
```

SEE ALSO

convert_from_polygon(2)
boolean_polygons(2)
resize_polygon(2)

convert_wire_ends

Converts the currently opened cell's signal wire ends.

SYNTAX

```
status_value convert_wire_ends
```

ARGUMENTS

no arguments

DESCRIPTION

This command searches the currently opened cell for signal wires having zero length extensions. Each of these wires is converted into a wire having half width extensions. If the wire is not convertible (i.e., the wire length is less than the width), the wire is deleted.

This command only modifies wires having a signal route type.

The command returns a status indicating success or failure.

EXAMPLES

The following example converts the "top" cell's zero length extension wires into half width extension wires.

```
prompt > open_mw_lib myLib
{myLib}
prompt > open_mw_cel top
{top}
prompt > convert_wire_ends
```

```
Warning: Removed net shape (VWIRE#1796726). (MWUI-140)
Warning: Removed net shape (VWIRE#1687148). (MWUI-140)
Warning: Removed net shape (VWIRE#1687150). (MWUI-140)
1
prompt > save_mw_cel
Information: Saved design named top.CEL;1. (UIG-5)
1
prompt > close_mw_cel
1
prompt > close_mw_lib
1
```

copy_collection

Duplicates the contents of a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection copy_collection collection1

collection collection1
```

ARGUMENTS

collection1

Specifies the collection to be copied. If the empty string is used for the *collection1* argument, the command returns the empty string (a copy of the empty collection is the empty collection).

DESCRIPTION

The **copy_collection** command is an efficient mechanism for creating a duplicate of an existing collection. It is more efficient, and almost always sufficient, to simply have more than one variable referencing the same collection. For example, if you create a collection and save a reference to it in a variable **c1**, assigning the value of **c1** to another variable **c2** simply creates a second reference to the same collection:

```
prompt> set c1 [get_cells "U1*"]
{"U1", "U10", "U11", "U12"}
prompt> set c2 $c1
{"U1", "U10", "U11", "U12"}
```

This has not copied the collection. There are now two references to the same collection. If you change the *c1* variable, *c2* continues to reference the same collection:

```
prompt> set c1 [get_cells "block1"]
{"block1"}
```

```
prompt> query_objects $c2
{"U1", "U10", "U11", "U12"}
```

There may be instances when you really do need a copy, and in those cases, **copy_collection** is used to create a new collection that is a duplicate of the original.

EXAMPLES

The following example shows the result of copying a collection. Functionally, it is not much different that having multiple references to the same collection.

```
prompt> set c1 [get_cells "U1*"]
{"U1", "U10", "U11", "U12"}
prompt> set c2 [copy_collection $c1]
{"U1", "U10", "U11", "U12"}
prompt> unset c1
prompt> query_objects $c2
{"U1", "U10", "U11", "U12"}
```

SEE ALSO

collections(2)

copy_mw_cel

Copies Milkyway designs from a source design library to a target design library.

SYNTAX

```
status copy_mw_cel
   -from source_mw_cel_name
   [-to target_mw_cel_name]
   [-from_library source_library_name]
   [-to_library target_library_name]
   [-hierarchy]
   [-check_only]
   [-overwrite]
```

Data Types

```
   source_mw_cel_name      collection
   target_mw_cel_name      string
   source_library_name     string
   target_library_name     string
```

ARGUMENTS

-from *source_mw_cel_name*

Specifies the Milkyway designs to be copied. If you use the **-hierarchy** option, you can specify only a single design. If you do not use the **-hierarchy** option, you can specify a single design or multiple designs.

You can specify a Milkyway design by name, pattern, or collection. For example, top matches a Milkyway design named top in the current design library; top* matches all Milkyway designs whose names start with top; and * specifies all Milkyway designs in the current design library.

If you specify a version in the source file name, only that version of the Milkyway design is copied. If you do not specify a version in the source file name, the latest version of the Milkyway design is copied.

-to *target_mw_cel_name*

Specifies the name of the target Milkyway design.

If you specify multiple source Milkyway designs in the **-from** option, the tool assumes that you want to

copy them without changing the names, so it ignores this option.

The option **-to** and option **-hierarchy** are mutually exclusive. You can specify only one of these options.

If you do not specify this option, the source Milkyway designs are copied to the target library without changing the name.

-from_library ***source_library_name***

Specifies the design library that contains the source Milkyway designs.

If you use patterns, such as "top" or "*", to specify the source Milkyway designs in the **-from** option, the tool ignores this option. All patterns are matched in the current design library.

If you do not specify this option, the tool uses the current design library as the source library.

-to_library ***target_library_name***

Specifies the target library to which the Milkyway designs are copied. Generally, you do not need to open the target library before running this command.

If you use both this option and the **-hierarchy** option, and the specified target design library does not exist, the tool creates a new library and copies the hierarchical design into it.

If you do not specify this option, the tool uses the current design library as the target library.

-hierarchy

Copies the source Milkyway design and its subdesigns to the target design library.

The option **-hierarchy** and option **-to** are mutually exclusive. You can specify only one of these options.

-check_only

Prints the list of Milkyway designs, views, and versions but does not copy the designs. This option is valid only when used with the **-hierarchy** option.

-overwrite

Overwrites the target Milkyway design when copying. This option cannot be used with the **-hierarchy** option.

DESCRIPTION

This command copies the specified Milkyway designs from the source design library to the target design library. You must have write permission for the target design library. Only read permission is required for the source design library.

You can use the **-hierarchy** option to copy both the top-level design and its subdesigns. Subdesigns that are instantiated from the reference library stay in the reference library and are not copied to the target design library. The target library sets the reference path to those reference libraries. The latest version of

any open and unsaved Milkyway design in the hierarchical tree is saved from memory into the target design library. If the open Milkyway design is not the latest version, the Milkyway design is copied from disk instead of memory. If the Milkyway design is not open, the Milkyway design on disk is copied.

The copied Milkyway designs include all information from the original Milkyway design. The copied Milkyway designs have their versions set to 1 if the target library is a new library. If the target library is not a new library, the version of the copied Milkyway design is incremented by 1 from the latest version before the copy, unless you use the **-overwrite** option.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example copies all Milkyway designs from the current library to another library named new_lib.

```
prompt> copy_mw_cel -from * -to_library new_lib
1
```

The following example copies the design named ORCA and all of its subdesigns from the current library to a library named lib1.

```
prompt> copy_mw_cel -hierarchy -to_library lib1 -from ORCA
Copied Library Files
From: /orca/orca_lib
To: lib1

Copied Cells
From: /orca/orca_lib
To: lib1

Copying cell: ORCA.CEL;1
Copying cell: BLENDER_2;
1
```

SEE ALSO

close_mw_cel(2)
create_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
remove_mw_cel(2)
rename_mw_cel(2)
mw_support_hier_fill_view(3)

copy_mw_lib

Copies a Milkyway library to another location.

SYNTAX

```
status copy_mw_lib
   [-from mw_lib]
   -to lib_name
```

Data Types

```
   mw_lib       string
   lib_name     string
```

ARGUMENTS

-from *mw_lib*

Specifies the name of the source Milkyway library to be copied. The *mw_lib* value can be a library name or a collection of libraries. By default, the command uses the current Milkyway library.

-to *lib_name*

Specifies the destination Milkyway library name.

DESCRIPTION

This command copies a Milkyway library to another location. It returns a status indicating success or failure.

EXAMPLES

The following example copies a Milkyway library design to another location.

```
prompt> copy_mw_lib -from design -to design.bak
1
```

SEE ALSO

rename_mw_lib(2)

cputime

Returns the CPU time, in seconds, of this run.

SYNTAX

cputime

ARGUMENTS

DESCRIPTION

This command returns the CPU time, in seconds, of the process in which it is running.

EXAMPLES

```
pt_shell> set x [cputime]
```

create_base_array

Creates a base array record in the design.

SYNTAX

```
status_value create_base_array
   [-tile_name tile_name]
   -coordinate rectangle
   [-direction direction]

tile_name string
rectangle list
direction string
```

ARGUMENTS

-tile_name *tile_name*

Specifies the name of unit tile used in the base array record.

If **-tile_name** is not specifies, by default it creates a base array using "unit" as its unit tile.

-coordinate *rectangle*

Specifies the lower left corner and the upper right corner of the base array. The values are specified in microns relative to the chip origin.

-direction *direction*

Specifies the direction of the base array record.

The validated values of direction can be: horizontal and vertical.

By default, the direction is set to horizontal.

DESCRIPTION

The **create_base_array** command allows you to create a base array record into database.

This command returns *1* if succeeds.

EXAMPLES

The following example creates a base array record.

```
prompt> create_base_array -coordinate {{200 200} {1800 1800}} -direction horizontal
-tile_name unit
1
```

SEE ALSO

```
report_base_arrays(2)
remove_base_arrays(2)
```

create_boundary

Create boundary for design.

SYNTAX

```
status create_boundary
   [-coordinate rectangle | -poly { point point ... }
     | -by_terminal]
   [-core]
   [-left_offset]
   [-right_offset]
   [-top_offset]
   [-bottom_offset]
   [-lib_cell_type type | design]
```

ARGUMENTS

-coordinate *rectangle*

Create boundary by given bounding box. This option is exclusive with **-poly** and **-by_terminal**, you can only use one of them. If all of them are omitted, it will create boundary according to current boundary. This option is also exclusive with **-lib_cell_type**.

-poly *{ point point ... }*

Specifies a rectilinear boundary by n points. This option is exclusive with **-coordinate** and **-by_terminal**, you can only use one of them. If all of them are omitted, it will create boundary according to current boundary. This option is also exclusive with **-lib_cell_type**.

-by_terminal

Create boundary based on the bounding box of all the terminals in the design. If **-lib_cell_type** option is also specified, then it will create boundary for lib_cells based on bounding box of all the terminals in the lib_cell.

-core

Indicate to use the coordinates as the core area.

-left_offset *left_offset*

Specifies the distance to adjust for left of the boundary.

-right_offset ***right_offset***

Specifies the distance to adjust for right of the boundary.

-top_offset ***top_offset***

Specifies the distance to adjust for top of the boundary.

-bottom_offset ***bottom_offset***

Specifies the distance to adjust for bottom of the boundary.

-lib_cell_type ***type***

Specifies the type of lib_cell to create boundary. This option is exclusive with **design** option. It also exclusive with **-coordinate** and **-poly**. If **-by_terminal** is also specified, it will create boundary according to terminal location, or else, it will create boundary according to current boundary.

design

Specifies the design for which to create boundary. This option is exclusive with **-lib_cell_type**. If both are omitted, it will create boundary for current design.

DESCRIPTION

The **create_boundary** command allows you to create boundary for design or lib_cell. If **-lib_cell_type** is given, then it will create boundary for lib_cells with specified type. Or else, if **design** is given, it will create boundary for the specified design. If neither **-lib_cell_type** nor **design** are specified, it will create boundary for current design.

You can specifies the boundary by **-coordinate** or by **-poly**. However, these two options are only avaiable for design boundary creation. You can not use them to create boundary for lib_cell.

Alternatively, you can specifies the boundary by **-by_terminal**. By using this option, it will create boundary based on all terminal's location.

If neither of the three option, **-coordinate**, **-poly**, **-by_terminal** are given, it will create boundary according to current boundary.

You can also use **-left_offset**, **-right_offset**, **-top_offset**, **-bottom_offset** to adjust the boundary. However, these four options can not be used with **-poly**.

EXAMPLES

The following command create boundary for current design by specifying a bounding box directly.

```
prompt> create_boundary -coordinate {{0 0} {100 100}}
1
```

The following command create boundary by specifying a point array.

```
prompt> create_boundary -poly {{0 0} {100 0} {100 100} {50 100} {50 50} {0 50}}
1
```

The following command create boundary by terminal location and a left offset.

```
prompt> create_boundary -by_terminal -left_offset 10
1
```

The following command create boundary for all io_pad reference library.

```
prompt> create_boundary -lib_cell_type io_pad
1
```

SEE ALSO

create_cell

Creates a cell in the current design.

SYNTAX

```
int create_cell
   [-from_library library]
   [-rotation {90 | 180 | 270}]
   [-mirror {x | y}]
   [-ignore_eco]
   [-without_check_status]
   -origin {x y}
   -from_design from_design
   cell_name

   string cell_name
   string from_design
   string from_library
```

ARGUMENTS

-from_design *design_name*

Specifies the design from which you want to create a cell.

-from_library *library*

Specifies the library containing the design from which you want to create the new cell. If not specified, *current library* will be used.

The *library* can be specified by absolute path, or a path related to the working directory of the application. Take an example, if the working directory is /A/B, and the library location is /A/C/my_library, you can specify the path as "/A/C/my_library", or "../C/my_library".

If the *library* is one existing reference library of the current library, it can be omitted here.

-rotation { 90 | 180 | 270}

Specifies the degree to which you want the cell rotated in freference to the master design. Valid values are: "90", "180", "270". If this option isn't specified, this command doesn't rotate the cell.

-mirror {x | y}

Specifies the orientation of the cell. Valid values are: "x", "y". If this option isn't specified, this command doesn't flip the cell.

When "x" is specified, the cell will be flipped on the Y axis(in the x direction); when "y" is specified, the cell will be flipped on the X axis(in the y direction). If nothing specified, the cell won't be flipped.

E.g. If you specify "90" as rotation degree, "x" as orientation, the cell will be rotated 90 degree at first, and then flipped on Y axis.

Note that there are multiple sets of *rotation* and *orientation* that may be used to obtain the same result. E.g. (90, x) is equal to (270, y).

-ignore_eco

Indicates this cell would be ignored during ECO operation.

The "-ignore_eco" and "-without_check_status" options are mutually exclusive.

-without_check_status

Indicates this cell wouldn't be ignored during ECO operation anyway.

The "-without_check_status" and "-ignore_eco" are mutually exclusive.

-origin {x y}

Specifies the original point of the new cell.

cell_name

Specifies the name of the cell you want to create.

DESCRIPTION

Creates a new cell from the specified design(cell master).

EXAMPLES

The following example creates a new cell.

```
mw_shell> create_cell -from_design OR2T\
-from_library library -origin {3500.0 4500.0}\
-relative 180 or2t_1
1
```

SEE ALSO

get_cells(2)
replace_cell(2)
remove_cell(2)

create_mw_bound

Creates a fixed move bound or floating group bound in the design.

SYNTAX

```
bound create_mw_bound
   [-auto | -bounding_box rectangle | -poly { point point ... }
     | -dimension {width height} ]
   [-name bound_name [-attached_instances instances]
     | -logical_name logic_name]
   [-effort low | medium | high | ultra]
   [-utilization util_value]
   [-type soft | hard]
   [-color bound_color]
   [-rigidity bound_rigidity]
   [-exclusive]
   [-fixed]
   [cell_list]
```

ARGUMENTS

-auto

Creates a move bound for the top-level logic node that is specified using the **-logical_name** option. The specified logical name is the bound name. If **-logical_name** is not used, by default it creates an empty move bound. You can add cells to this move bound later by using the **update_mw_bound** command.

-bounding_box *rectangle*

Specifies a particular bounding box to represent the move bound. If a bounding box not given, a bounding box is assigned to the move bound according to the **-attached_instances** or **-logical_name** and **-utilization** options.

-poly *{ point point ... }*

Specifies the coordinates of *n* points of the new rectilinear shape for the move bound.

-dimension *{ width height }*

Specifies the dimension of a group bound.

-name *bound_name*

Specifies the name of the bound to be created. The hierarchy delimiter cannot be part of the bound name. If neither **-name** nor **-logical_name** are specified, an internal name is automatically generated. This option and **-logical_name** are mutually exclusive.

-attached_instance *instances*

Specify a list of instances to be attached to the move bound. This option can work with other options: -bounding_box, -poly, and -name, but it can not work with -logical_name.

-logical_name *logic_name*

Specifies the name of the logical node that represents a corresponding soft region of the logical node under the top-level design and is to be mapped into a list of corresponding physical instances. The move bound is created based on the location and shapes of those physical instances. Those physical instances are attached to the newly created move bound and the name is inherited by the *logic_name*.

This option cannot be used with **-name**, because **-name** is the user-defined name's bound instead of the name of the logical node.

-effort low | medium | high | ultra

Specifies the effort to bring cells closer inside a group bound. The default effort is **medium**.

-utilization *util_value*

Specifies a float number for the utilization fraction of a logical node or the whole chip. The *util_value* must be within [0.1, 1.0]. The default value is 0.7.

-type soft | hard

Specifies the type of the bound. Valid values are **soft** and **hard**. The default is **soft**.

-color *bound_color*

Specifies the colors that are used to draw the move bound and its associated instances. Accepted *bound_color* values are a color ID of 0 through 63, or the name of the color.

-rigidity *bound_rigidity*

Specifies how hard the placement engine tries to keep the cells inside the move bound boundary. The value must within [1, 10], where 10 is the most rigid or hard constraint of the move bound, and 1 is the least rigid. When the value is smaller than 10, the placement constraints are relaxed and placement can be optimized based on connectivity. This property is used by the Astro placement engine and has no effect on **fphPlaceDesign**. The default value is 10.

-exclusive

Assigns a property type of **exclusive** to the move bound being created. Move bounds that are **exclusive** require all of their cells to be placed inside them and prohibit the placement of other cells in the same area. You must set the move bound to **exclusive** to legalize the placement and commit the move bound to physical blocks. This property is used by **fphPlaceDesign**.

-fixed

Assigns a property type of **fixed** to fix the location of the move bound.

cell_list

Specifies a list of cells to be included in the group bound.

DESCRIPTION

The **create_mw_bound** command allows you to define region-based placement constraints for coarse placement. There are two types of bounds available: move bounds and group bounds. Move bounds restrict the placement of cells to a specific region of the core area. Move bounds require absolute coordinates to be specified, using the **-auto**, **-bounding_box**, or **-poly** option. Group bounds are floating region constraints. Cells in the same group bound are placed within a specified bound, but the absolute coordinates are not fixed. Instead, the coordinates are optimized by the placer. Usually, **-dimension** is specified for a group bound.

If **-auto**, **-bounding_box**, or **-poly** is specified, a move bound is created. If **-dimension** is specified, a group bound with the given dimension is created. If non of above switches is used, a group bound is created with a bounding box computed internally by the tool. In this case, you can use **-effort** to specify the effort level to use to bring the cells closer. You cannot use **-effort** with **-auto**, **-bounding_box**, **-poly**, or **-dimension**.

All automatically generated bounds are soft bounds. You cannot use **-type** to change the bound type of these automatically group bounds.

EXAMPLES

The following example creates a move bound corresponding to the muxA logical node:

```
prompt> create_mw_bound -auto -logical_name muxA
{"muxA"}
```

The following example creates a group bound:

```
prompt> create_mw_bound -dimension {10 10} -name muxB [get_cell and*]
```

SEE ALSO

```
get_bounds(2)
remove_mw_bounds(2)
update_mw_bound(2)
```

create_mw_cel

Creates a Milkyway desgin.

SYNTAX

```
status_value create_mw_cel
   [-view CEL | FRAM | FILL | err ]
   [-verbose]
   mw_cel_name
```

Data Types

```
   mw_cel_name  string
```

ARGUMENTS

-view CEL | FRAM | FILL | err

Specifies the view name of the Milkyway design you want to create. The **CEL**, **FRAM**, **FILL**, and **err** arguments are mutually exclusive. By default, the command creates a **CEL** view Milkyway desgin.

-verbose

Prints additional messages.

mw_cel_name

Specifies the name of the Milkyway design to create. The value of *mw_cel_name* must be unique within the current library. The length of a Milkyway desgin name must be less than 127. Do not specify a view name or version in the *mw_cel name*. For example, *test.cel* or *test.cel;2* are not valid values for *mw_cel name*.

DESCRIPTION

This command creates a new Milkyway design. It automatically opens this newly-created Milkyway design

and sets it as the current Milkyway design.

EXAMPLES

The following example creates a Milkyway design named *top*.

```
prompt> create_mw_cel top
{"top"}


prompt> current_mw_cel
{"top"}
```

SEE ALSO

close_mw_cel(2)
copy_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
remove_mw_cel(2)
rename_mw_cel(2)
save_mw_cel(2)

create_mw_lib

Creates a Milkyway library.

SYNTAX

```
status create_mw_lib
   [-technology technology_file_name]
   [-plib plib_file_name]
   [-hier_separator sep]
   [-bus_naming_style style]
   [-mw_reference_library lib_list]
   [-reference_control_file rc_file_name]
   [-open]
   libName
```

Data Types

```
   technology_file_name     string
   plib_file_name           string
   sep                      character
   style                    string
   lib_list                 string
   rc_file_name             string
   libName                  string
```

ARGUMENTS

-technology *technology_file_name*

Specifies the name of the technology file for the newly created Milkyway library. This option and **-plib** are mutually exclusive.

-plib *plib_file_name*

Specifies the name of the plib file for the newly created Milkyway library. Valid file is of .plib format or .pdb format. This option and **-technology** are mutually exclusive. If routing directions in the .plib file are used to set preferred routing directions for the design library, you must specify the reference libraries by using the **-reference_control_file** or **-mw_reference_library** option.

-hier_separator *sep*

Specifies the character to be used as a separator in hierarchical names. The default hierarchical

separator is slash (/).

-bus_naming_style *style*

Specifies the bus naming style for the library. The default bus naming style is [%d].

-mw_reference_library *lib_list*

Specifies a list of reference libraries to be used for the new library.

-reference_control_file *rc_file_name*

Specifies the reference control file used to set reference library information for the new library.

-open

Opens the library after creation.

libName

Specifies the name of the Milkyway library to be created.

DESCRIPTION

This command creates a Milkyway library.

The **-technology** and **-plib** options are mutually exclusive, and at least one of them must be specified.

Some advanced technologies require more than the default 255 layers. In these cases, you can extend the number of layers to 4095 by executing the **extend_mw_layers** command before you exeucte the **create_mw_lib** command. Once created in the default 255-layer mode or extended 4095-layer mode, the Milkyway library must remain in that layer mode. For details, see the man page for the **extend_mw_layers** command.

By default, the newly created Milkyway library is not open in the current session. To manipulate it, first run the **open_mw_lib** command. The library can also be opened by using the **-open** option, but to makes the scripts more reusable, you should use **open_mw_lib**.

All strings stored in this library, as well as in designs belonging to it, are case-sensitive, and all string operations are performed as case-sensitive.

A flag to indicate success or failure is returned.

EXAMPLES

The following example creates a Milkyway design library with a technology file and bus naming style as [%d].

```
prompt> create_mw_lib design -technology test.tf \
-bus_naming_style {[%d]}
1
```

SEE ALSO

close_mw_lib(2)
extend_mw_layers(2)
open_mw_lib(2)

create_net

Creates nets in the specified design.

SYNTAX

```
int create_net
   [-design design]
   net_list

design list


net_list list
```

ARGUMENTS

-design *design*

Specifies the design where the net is to be created. It can be a name or collection. By default, the command uses the current design.

net_list

Specifies the names of the nets to be created in the design. Each net name must be unique.

DESCRIPTION

This command creates new net objects in the design. It creates only scalar (single-bit) nets.

Nets connect pins and ports in a design. When the **create_net** command creates nets, they are not connected. To establish this connection, you can use the **connect_net** command.

You can use the **remove_net** command to remove nets from the design.

EXAMPLES

The following example creates net objects named *N1*, *N2*, *N3*, and *N4*.

```
prompt> current_design

prompt> create_net {N1, N2, N3, N4}

prompt> get_nets "*"
```

SEE ALSO

```
all_connected(2)
connect_net(2)
current_design(2)
disconnect_net(2)
get_nets(2)
remove_net(2)
```

create_net_shape

Creates a new net shape and adds it to a net in the current design.The command returns a collection handle pointing to the created shape (or 0 if the command fails).

SYNTAX

```
collection create_net_shape
   [-type wire | path | rect | poly]
   {-bbox {{ll_x ll_y} {ur_x ur_y}}}
    | {-origin origin [-length length] [-width width]}
    | {-points list_of_points}
    | {-boundary boundary}
   [-path_type {square | round | extend_half_width | octagon | 0 | 1 | 2 | 3}]
   [-layer layer]
   [-net net_name]
   [-vertical]
   [-route_type route_type]
   [-datatype int_range]
   [-avoid_short_segment]

   string origin
   int length
   int width
   string layer
   list net_name
   list    boundary
   list    list_of_points
   string  route_type
   int     int_range
```

ARGUMENTS

-type *wire* | *path* | *rect* | *poly*

Specifies the net shape type.

If you do not specify this option, the default net shape depends on how you specify the net shape. The net shape type is determined by using the following rules, in order of precedence:

1. If you use the **-origin** option, the net shape is **wire**. The wire is horizontal, unless you also specify the **-vertical** option.

2. If you use the **-bbox** option, the net shape is **wire**.

3. If you use the **-points** option, the net shape is **path**.

4. If you use the **-boundary** option, the net shape is **poly**.

-bbox ***{{ll_x ll_y} {ur_x ur_y}}***

Specifies a rectangular defining the shape. The bounding_boxs of the rectangle are relative to the origin of the current design.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-origin ***origin***

Specifies the point where you want to place this wire net shape.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-points ***rectilinear_point_sequence***

Specifies the point sequence of the net shape for paths. Support horizontal, vertical and 45 degree path.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-boundary ***boundary***

Specifies the boundary of the net shape. You must specify at least 4 rectilinear points.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-length ***length***

Specifies the length of the wire net shape.

This argument should be used with "-origin".

-width ***width***

Specifies the width of the wire to create.

You can set a default width value with a Tcl variable named *mw_wire_width*. E.g. "set mw_wire_width 2" sets the default value as 2. So that if you don't specify this option when creating a wire net shape, this default width will be used.

When both *mw_wire_width* and *-width* are specified, the latter is used.

-path_type ***path_type***

Specifies the alignment type of a wire or path end. Possible values are:

1) "0" or "square" square, no extension;

2) "1" or "round" round, half width extension;

3) "2" or "extend_half_width" square, half width extension;

4) "3" or "octagon" octagon, half width extension. Default value is "0" or "square".

Also there is a Tcl variable named "mw_wire_path_type" used as the default path type when "-path_type" isn't specified.

-layer *layer*

Specifies the layer for the net shape. You can specify the layer by using the layer name from the technology file or by using the **get_layers** command.

Also there is a Tcl variable name "mw_wire_layer" used as the defualt layer name when *-layer* isn't specified.

-net *net_name*

Specifies a net to add a wire net shape on. It can be a power net or a regular net. You can specify it by net name, or a net collection handler which contains one net.

-vertical

Indicates the orientaiton of the wire net shape you want to create. Without specifying this option, a horizontal wire net shape will be created.

-route_type *route_type*

Specifies the route type of the net shape. Valid values for this option are: user_enter User entered signal_route, signal_route_detail Detail routing signal_route_global Global routing pg_ring Power or ground (PG) ring pg_strap PG strap pg_macro_io_pin_conn PG net that connects to the PG pin of an I/O pad cell or a macro cell pg_std_cell_pin_conn PG net that connects to the PG pin of a standard cell clk_ring Clock ring clk_strap Clock strap clk_zero_skew_route Clock zero skew route bus Bus shield, shield_fixed Fixed shield shield_dynamic Dynamic shield fill_track, clk_fill_track Fill track

By default, the route type is signal_route.

-datatype *int_range*

Specifies the data type number

By default, the data type is 0.

-avoid_short_segment

Specifies that segments shorter than a half width should be avoided for paths.

DESCRIPTION

This command creates a shape object that is attached to a net and returns a collection containing the created object.

The valid shape types are: Wire (horizontal or vertical) Path Rectangle Polygon

For convenience of usage, there are three Tcl variable defined as defaul value which are necessary when creating wire net shape. They are: mw_wire_layer mw_wire_width mw_wire_path_type You can use

printvar to view their value, and call *set variable_name value* to give a new value.

EXAMPLES

The following example create a wire net shape with a bounding box.

```
prompt>create_net_shape -layer M1R \
? -bbox {{35 67} {89 69}} \
? -path_type 2
{"HW4782"}
```

The following example create a wire net shape with default settings.

```
prompt>printvar mw_wire*
mw_wire_layer      = "NULL"
mw_wire_path_type  = "0"
mw_wire_width      = "0"

prompt>set mw_wire_layer M1R
M1R

prompt>set mw_wire_width 4
4

prompt>set mw_wire_path_type 2
2

prompt>create_net_shape -bbox \
? {{5 7} {7 29}} -vertical
{"HW4783"}
```

The following example create a wire net shape with specifying start point and length, and connect to a specified net.

```
prompt> create_net_shape -origin {5 5}\
?  -leng 30  -net n300
{"HW3690"}
```

SEE ALSO

remove_net_shape(2)
get_net_shapes(2)

create_placement_blockage

Creates a blockage (prohibited) area for placement on the specified region.

SYNTAX

```
collection_handle create_placement_blockage
   [-type hard | soft | pin | hard_macro]
   [-name string]
   -coordinate {{ll_x ll_y} {ur_x ur_y}}

ll_x string
ll_y string
ur_x string
ur_y string
```

ARGUMENTS

-type hard | soft | pin | hard_macro

Specifies the type of the placement blockage to create.

1) soft A soft placement blockage allows buffers or inverters to be inserted within the blockage during netlist optimization commands such as In-Placement Optimization or Post-Placement Optimization. A soft placement blockage is on layer 222.

2) hard The command will not place a cell in a hard placement blockage area. A hard placement blockage is on layer 221;

3) pin With a pin blockage, during pin assignment the global router does not route in the specified area and the pin placer does not assign pins in the specified area. A pin blockage is on layer 228.

4) hard_macro With a hard_macro blockage, the placer places standard cells but not hard macros in the specified region. A hard macro blockage is on layer 199.

The **hard**, **soft**, **pin** and **hard_macro** arguments are mutually exclusive. By default, the command creates a **hard** view design.

-name *string*

Specify a name for this new created placement blockage. To be compatible, a legal name shouldn't begin with a "pb".

The given name should be unique in that of existing placement blockages. If not, this command display a warning message and stop execution.

If no name given, this command returns a run-time fake name which is formed as a prefix "pb" along with its object ID. For example, "pb7789" is a run-time name.

-coordinate {{*ll_x ll_y*} {*ur_x ur_y*}}

Specifies a rectangular area in which to create the placement blockage. The rectangle coordinates are relative to the current design and identify the lower-left and upper-right corner of the rectangular area.

DESCRIPTION

This command enables you to create placement blockages for placement on the specified region.

EXAMPLES

The following example creates a blockage for placement.

```
prompt> create_placement_blockage -coordinate {{2 2} {25 25}}
{"pb7789"}
```

SEE ALSO

create_route_guide(2)
get_placement_blockages(2)
get_route_guides(2)
remove_placement_blockage(2)
remove_route_guide(2)

create_port

Creates a top level port.

SYNTAX

```
collection_handle create_port
   [-direction {in | out | inout | tristate} ]
   port_name
   -net net_name

net_name string


port_name string
```

ARGUMENTS

-direction {in | out | inout | tristate}

Specifies the direction of the port to create. The **input**, **output**, and **tristate** arguments are mutually exclusive. The braces {} are included only for readability; they are not part of the syntax. By default, the command creates an **input** port.

port_name

Specifies the name of the port to create.

-net *net_name*

Specifies the net to which the new port connects.

DESCRIPTION

This command creates a top-level port on a design. If the port you specified already exists for the specified net, the command displays an error message.

EXAMPLES

The following example creates ports named *data1* and *data2* connected to a net named *n300*.

```
prompt> get_ports *
{"VDD", "VSS"}


prompt> create_port data1 -net n300
{"data1"}


prompt> create_port data2 -net n300
{"data2"}


prompt> get_ports *
{"VDD", "VSS", "data1", "data2"}
```

SEE ALSO

```
get_ports(2)
remove_port(2)
```

create_route_guide

Creates a route guide (prohibited)area for routing at the specified region.

SYNTAX

```
collection_handle create_route_guide
   [-no_signal_layers {layer_list}]
   [-zero_min_spacing]
   [-no_preroute_layers {layer_list}]
   [-preferred_direction_only_layers {layer_list}]
   [-repair_as_single_sbox]
   [-horizontal_track_utilization percentage]
   [-vertical_track_utilization percentage]
   [-switch_preferred_direction]
   -coordinate {{ll_x ll_y} {ur_x ur_y}}

   list layer_list
   int percentage
```

ARGUMENTS

-coordinate *{{ll_x ll_y} {ur_x ur_y}}*

Specifies a rectanglular area in which to create the route guide. The rectangle coordinates are relative to the current design.

-no_signal_layers {layer_list}

Specifies a name list of the layers on which signal routing isn't allowed.

-zero_min_spacing

Allow zero minimal spacing.

This option is only effective when the option "-no_signal_layers" is specified.

-no_preroute_layers {layer_list}

Specifies a name list of the layers on which automatic prerouting isn't allowed.

-preferred_direction_only_layers {layer_list}

Specifies the layers that cannot make nonPreferredDirection wire.

-repair_as_single_sbox

Indicates to route the specified area as one switch box when running the related search repair command, and when there are violations inside the area.

Otherwise to partition the area into several switch boxes and to route one switch box at a time.

You are suggested to specify this option if there is a difficult violation on a prerouted wire or inside a large macro.

-horizontal_track_utilization percentage

Specifies the horizontal utilization for global routing within the guide area.

-vertical_track_utilization percentage

Specifies the vertical utilization for global routing within the guide area.

-switch_preferred_direction

Changes the horizontal direction to vertical and the vertical direction to horizontal.

You can specify this option when you want to route long and narrow channels between two macros with metal1 and metal3, instead of metal2, in the "long" direction.

-name *string*

Specify a name for this new created route guide. To be compatible, a legal name shouldn't begin with a "rg".

The given name should be unique in that of existing route guide. If not, this command display a warning message and stop execution.

If no name given, this command returns a run-time fake name which is formed as a prefix "rg" along with its object ID. For example, "rg7789" is a run-time name.

DESCRIPTION

This command enables you to create route guide for routing. A route guide prevent routing from being placed within the specified coordinates and on a given layer. These are regions those must not be used by automatic routing commands or port assignment.

EXAMPLES

The following example creates a route guide with specifying there shouldn't be signal routing within this guide area on two layers: m3 and m20.

```
prompt> create_route_guide -no_signal_layer {m3 m20} -preferred_direction_only_layers {m4} -coordin
```

```
{"RG7648"}
```

SEE ALSO

create_placement_blockage(2)
remove_route_guide(2)
remove_placement_blockage(2)
get_placement_blockages(2)
get_route_guides(2)

create_routing_blockage

Creates a new routing blockage on metal or via routing blockage layers.

SYNTAX

```
collection create_routing_blockage
   -layers layer_list
   -bbox {bounding_box} | -boundary {polygon_boundary_points}
```

Data Types

```
   layer_list              list of layers
   bounding_box            lower left and upper right coordinates
   polygon_boundary_points  list of coordinates
```

ARGUMENTS

-layers *layer_list*

Specifies the layers in which the routing blockage is located. If you specify more than one layer, the tool creates multiple routing blockages simultaneously.

The valid value are: metal1Blockage-metal15Blockage, via1Blockage-via14Blockage, polyBlockage, and polyContBlockage.

This is a required option.

-bbox {*bounding_box*}

Specifies the bounding box of a rectangular routing blockage. The format of the bounding box specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

The unit for the coordinates is the unit specified in the technology file.

This option and the **-boundary** option are mutually exclusive. You must specify one of these options.

-boundary *polygon_boundary_points*

Specifies the boundary points of a rectilinear routing blockage. The format of the boundary specification is {{*x1 y1*} {*x2 y2*} ... {*xn yn*}}.

The unit for the coordinates is the unit specified in the technology file.

This option and the **-bbox** option are mutually exclusive. You must specify one of these options.

DESCRIPTION

The **create_routing_blockage** command creates metal or via routing blockages on the metal, via, or poly blockage layers. Unlike route guides, which direct the router to change routing information on nets that go through a route guide, routing blockages direct the router to avoid routing through these areas.

.prod icc

By default, Zroute does not honor the routing blockages created by this command. To have Zroute honor these routing blockages, you must run the **set_route_zrt_common_options -read_user_metal_blockage_layer true** command before performing routing. .prod all

You can create either rectangular routing blockages by using the **-bbox** option or rectilinear routing blockages by using the **-boundary** option.

When the tool creates routing blockages, it automatically assigns a name of RB_*object_id* to each routing blockage.

EXAMPLES

The following example creates a rectangular via routing blockage in the current design.

```
prompt> create_routing_blockage -layers via5Blockage \
   -bbox {{0 0} {100 100}}
{RB_29440}
```

The following example creates two rectilinear metal routing blockages in the current design.

```
prompt> create_routing_blockage -layers {metal13Blockage via4Blockage} \
   -boundary {{0 0} {50 0} {50 50} {100 50} {100 100} {0 100}}
{RB_29696 RB_29697}
```

The following example creates a rectangular metal routing blockage in the current design on each metal blockage layer in the design library:

```
prompt> create_routing_blockage \
   -layers [get_layers -include_system -filter {name=~metal*Blockage}] \
   -bbox {{0 0} {50 50}}
{RB_29441 RB_29442 RB_29443 RB_29444 RB_29445 RB_29446 RB_29447
RB_29448 RB_29449 RB_29450 RB_29451 RB_29452 RB_29453 RB_29454 RB_29455}
```

SEE ALSO

```
get_layers(2)
get_routing_blockages(2)
remove_routing_blockage(2)
```

create_terminal

Creates a rectangular geometry and adds it to a port in the current design.The command returns a collection handle pointing to the created terminal (or 0 if the command fails).

SYNTAX

```
collection_handle create_terminal
   {-bbox rect | -boundary boundary}
   -layer layer
   -port string
   [-direction { left | right | up | down }]
   [-name string]
```

ARGUMENTS

-bbox *rect*

Specifies the bounding box of the terminal.

This option cannot be combined with the *-boundary* option.

-boundary *boundary*

Specifies the rectilinear boundary of the terminal.

This option cannot be combined with the *-bbox* option.

-layer *layer*

Specifies the layer of the terminal. The layer name should exist in the technology file.

-port *port_name*

Specifies the name of the port to which the terminal is added. It can be a power port or a regular port.

-direction *pin_direction*

Specifies a list of allowable access directions for the terminal.

Valid values are: left, right, up or down.

If not supplied no access direction is assumed.

-name *name*

Specifies the name of the terminal. It should be an unique terminal name in the current design.

If no name is specified then one is automatically generated from the port name.

Note: It is recommended that no name is specified, so the automatically generated name is used, as some operations require that the terminal name conforms to terminal naming conventions (See below).

If the specified terminal name does not match terminal naming conventions a warning will be issued.

DESCRIPTION

This command creates a terminal for the port on the current design.

EXAMPLES

The following example create a terminal for the specified port.

```
prompt> create_terminal -layer M2R -port port1 -bbox {{254 527} {256 530}}
{"port1_shape"}

prompt> create_terminal -layer M2R -port port1 -bbox {{254 532} {256 535}}
{"port1_shape 1"}
```

SEE ALSO

remove_terminal(2)
get_terminals(2)

create_text

Creates a text record at the specified region.

SYNTAX

```
collection_handle create_text
   [-layer layer]
   [-height height]
   [-orient orient]
   [-anchor anchor]
   -origin point
   text
```

Data Types

```
   layer   string
   height  float
   orient  string
   anchor  string
   point   list
   text    string
```

ARGUMENTS

-layer *layer*

Specifies the name or the number of the layer on which to create text.

The default layer is defined by the **mw_text_layer** Tcl variable. You can view its value by using **printvar** and set a new value for it by using **set**. By default, the command uses **mw_text_layer**.

-height *height*

Specifies the size of the new text in the *y* direction, specified in user units.

The default height is defined by the **mw_text_height** Tcl variable. You can view its value by using **printvar** and set a new value for it by using *set*. By default, the command uses **mw_text_height**.

-orient *orient*

Specifies the orientation you want for placement of the text. The following list shows the valid values for *orient*:

N
W
S
E
FN
FE
FS
FW
0
90
180
270
0-mirror
90-mirror
180-mirror
270-mirror

The default value of *orient* is **0**.

-anchor *anchor*

Specifies the placement of text around the origin. The following list shows the valid values for *anchor*:

lb
cb
rb
lc
c
rc
lt
ct
rt

The default value of *anchor* is *lb*.

-origin *point*

Specifies the coordinate where new text should be placed. The format of a point specification is {*x y*}.

text

Specifies the text string that should appear in the design.

DESCRIPTION

This command creates a text object for recording a text string in the design.

EXAMPLES

The following example creates a text record.

```
prompt> printvar mw_text_layer
mw_text_layer        = "NULL"

prompt> set mw_text_layer TOPMETAL
TOPMETAL

prompt> printvar mw_text_height
mw_text_height       = "0"

prompt> set mw_text_height 3.0
3.0

prompt> create_text -origin {10 10} helloworld
{"TEXT#6400"}
```

SEE ALSO

get_texts(2)
remove_text(2)

create_track

Creates tracks for a routing layer or poly layer.

SYNTAX

```
int create_track
   -layer layer
   [-space track_pitch]
   [-count number_of_tracks]
   [-coord start_x_or_y]
   [-dir X | Y]
   [-bounding_box track_boundary_box]
   [-width width]
   [-reserved_for_width true | false]
   [-mask_constraint constraint]
```

Data Types

```
   layer                 string
   track_pitch           float
   number_of_tracks      integer
   start_x_or_y          float
   track_boundary_box    rectangle
   width                 float
```

ARGUMENTS

-layer *layer*

Specifies the layer to use the tracks. You can specify the layer name, the layer number, or a collection containing one layer object.

-space *track_pitch*

Specifies the pitch between tracks. The pitch is the center-to-center distance between two routing wires in adjacent tracks. By default, this distance is the same as the routing pitch from the unit tile in the Milkyway reference library. The pitch unit size is specified in technology file, typically microns.

-count *number_of_tracks*

Specifies how many tracks to create. By default, the tracks use the entire die area. If you do not specify this option, the value is automatically calculated from the available area of the die or bounding

box.

-coord *start_x_or_y*

Specifies the x-coordinate or y-coordinate of the first track, depending on whether the tracks are vertical or horizontal, as determined by the **-dir** option. The unit size for the coordinate is specified in the technology file, typically microns. By default, the coordinate is the left or bottom coordinate of the bounding box, or one-half the pitch inside of the die area.

-dir X | Y

Specifies the direction in which the tracks are placed, either **X** (horizontal) or **Y** (vertical). By default, the direction is the preferred routing direction of the layer, which is specified by the unit tile in the Milkyway reference library.

-bounding_box *track_boundary_box*

Specifies the lower-left and upper-right coordinates of the bounding box that encloses all the tracks, in the form of {{*x1 y1*} {*x2 y2*}} The unit size for the coordinates is specified in the technology file, typically microns. By default, there is no bounding box and the tracks cover the available die area.

-width *width*

Specifies the width of wires associated with the tracks. By default, wires of any width can be used with these tracks. The width value must be greater than zero.

-reserved_for_width true | false

Specifies whether the tracks are reserved for routes created by nondefault routing (NDR) rules. When this option is set to **true**, the tracks can only be used for routing wires of the width specified by the **-width** option, and only by nondefault routing (NDR) rules. This option can be used only with the **-width** option.

-mask_constraint *constraint*

Specifies the mask constraint for the terminal.

The valid mask_constraints are **any_mask**, **mask1_soft**, **mask1_hard**, **mask2_soft**, **mask2_hard**, **same_mask**, **mask3_soft**, **mask3_hard**.

.prod all

DESCRIPTION

This command creates a group of tracks on the floorplan so the router can use them to perform detail routing, or the **insert_metal_filler** command can use them to fill the poly layer at the chip finishing stage.

You must specify either a routing layer or poly layer for the tracks. The option of creating tracks for a poly layer is intended for fill purposes only, not routing.

By default, the command creates tracks the preferred routing direction and fills the available die area, starting from one-half the pitch from the die corner. You can optionally specify the track direction (X or Y),

the coordinate of the first track, the number of tracks to create, or a bounding box to be filled with tracks.

The tracks created by this command are part of the design database. They are saved in the Milkyway database when you use the **save_mw_cel** command and in DEF format when you use the **write_def** command.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example creates routing tracks for a routing layer named "m3" on the floorplan and reports the newly created routing tracks.

```
prompt> create_track -layer m3
Warning: Space is not specified. Using pitch. (MWUI-124)
Warning: Direction is not specified. Using the layer preferred direction.
(MWUI-125)
Warning: Coordinate value is not specified. Defaulting to the left(bottom)
coordinate of the track's bounding box. (MWUI-126)
Warning: Count value is not specified. Covering the bounding box of track,
depending on its space value. (MWUI-127)
1
prompt> report_track -layer m3
 ...
Layer          Direction     Start      Tracks    Pitch     Attr
----------------------------------------------------------------
Attributes :
         usr : User defined
         def : DEF defined

m3                Y         0.280       3696      0.560     usr
1
```

The following example creates reserved tracks for a routing layer named "m2" with width of 0.06, using a nondefault routing rule, and reports the newly created tracks.

```
prompt> create_track -layer m2 -dir Y -coord 0.438 -space 0.57 \
  -width 0.06 -bounding_box [list "0.000 0.12" "190.260 341.88"] \
  -reserved_for_width true
Warning: Count value is not specified. Covering the bounding box of track,
depending on its space value. (MWUI-127)
1
prompt> report_track -layer m2
 ...
Layer          Direction     Start     Tracks    Pitch      Attr
----------------------------------------------------------------
Attributes :
         usr : User defined
         def : DEF defined

m2                Y         0.438      790      0.570      usr,width=0.060,reserved_for_width
1
```

SEE ALSO

```
get_tracks(2)
remove_track(2)
report_track(2)
```

create_user_shape

Creates a new user shape. A user shape is a metal shape that is not associated with a net.

SYNTAX

```
collection create_user_shape
   [-type wire | path | trap | rect | poly]
   {-bbox {{ll_x ll_y} {ur_x ur_y}}
    | {-origin origin [-length length] [-width width]}
    | {-points list_of_points}
    | {-boundary boundary}
   [-path_type {square | round | extend_half_width | octagon | 0 | 1 | 2 | 3}]
   [-layer layer]
   [-vertical]
   [-route_type route_type]
   [-datatype int]
   [-avoid_short_segment]

   string origin
   int length
   int width
   string layer
   list    boundary
   list    list_of_points
   string  route_type
   int     integer
```

ARGUMENTS

-type *wire* | *path* | *trap* | *rect* | *poly*

Specifies the type of the user shape. Valid values are wire, path, trap (trapezoid), rect (rectangle), and poly (polygon).

If you do not specify this option, the default user shape depends on how you specify the geometry of the user shape. The type is determined by using the following rules, in order of precedence:

1. If you use the **-origin** option, the user shape is **wire**. The wire is horizontal, unless you also specify the **-vertical** option.

2. If you use the **-bbox** option, the user shape is **rect**.

3. If you use the **-points** option, the user shape is **poly**.

4. If you use the **-boundary** option, the user shape is **poly**.

-bbox ***{{ll_x ll_y} {ur_x ur_y}}***

Specifies a rectangular defining the shape. The bounding_boxs of the rectangle are relative to the origin of the current design.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-origin ***origin***

Specifies the point where you want to place this wire user shape.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-points ***rectilinear_point_sequence***

Specifies the point sequence of the user shape for wire, path or trap. Support horizontal, vertical and 45 degree path.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-boundary ***boundary***

Specifies the boundary of the user shape. You must specify at least 4 rectilinear points.

The "-bbox", "-origin", "-points", "-boundary" are mutually exclusive. You must specify one of these options.

-length ***length***

Specifies the length of the wire user shape.

This argument should be used with "-origin".

-width ***width***

Specifies the width of the wire to create.

You can set a default width value with a Tcl variable named *mw_wire_width*. E.g. "set mw_wire_width 2" sets the default value as 2. So that if you don't specify this option when creating a wire user shape, this default width will be used.

When both *mw_wire_width* and *-width* are specified, the latter is used.

-path_type ***path_type***

Specifies the alignment type of a wire or path end. Possible values are:

1) "0" or "square" square, no extension;

2) "1" or "round" round, half width extension;

3) "2" or "extend_half_width" square, half width extension;

4) "3" or "octagon" octagon, half width extension. Default value is "0" or "square".

Also there is a Tcl variable named "mw_wire_path_type" used as the default path type when "-path_type" isn't specified.

-layer *layer*

Specifies the layer for the user shape. You can specify the layer by using the layer name from the technology file or by using the **get_layers** command.

Also there is a Tcl variable name "mw_wire_layer" used as the defualt layer name when *-layer* isn't specified.

-vertical

Indicates the orientaiton of the wire user shape you want to create. Without specifying this option, a horizontal wire user shape will be created.

-route_type *route_type*

Specifies the route type of the user shape. Valid values for this option are: user_enter User entered signal_route, signal_route_detail Detail routing signal_route_global Global routing pg_ring Power or ground (PG) ring pg_strap PG strap pg_macro_io_pin_conn PG net that connects to the PG pin of an I/O pad cell or a macro cell pg_std_cell_pin_conn PG net that connects to the PG pin of a standard cell clk_ring Clock ring clk_strap Clock strap clk_zero_skew_route Clock zero skew route bus Bus shield, shield_fixed Fixed shield shield_dynamic Dynamic shield fill_track, clk_fill_track Fill track

By default, the route type is signal_route.

-datatype *integer*

Specifies the data type number

By default, the data type is 0.

-avoid_short_segment

Specifies that segments shorter than a half width should be avoided for paths.

DESCRIPTION

This command creates a shape object that is not attached to a net and returns a collection containing the created object.

The valid shape types are: Wire (horizontal or vertical) Path Trapezoid Rectangle Polygon

For convenience of usage, there are three Tcl variable defined as defaul value which are necessary when creating wire user shape. They are: mw_wire_layer mw_wire_width mw_wire_path_type You can use *printvar* to view their value, and call *set variable_name value* to give a new value.

EXAMPLES

The following example create a wire user shape with a bounding box.

```
prompt>create_user_shape -type wire -layer M1R \
? -bbox {{35 67} {89 69}} \
? -path_type 2
{"HW4782"}
```

The following example create a wire user shape with default settings.

```
prompt>printvar mw_wire*
mw_wire_layer      = "NULL"
mw_wire_path_type  = "0"
mw_wire_width      = "0"

prompt>set mw_wire_layer M1R
M1R

prompt>set mw_wire_width 4
4

prompt>set mw_wire_path_type 2
2

prompt>create_user_shape -bbox -type wire\
? {{5 7} {7 29}} -vertical
{"HW4783"}
```

The following example create a wire user shape with specifying start point and length.

```
prompt> create_user_shape -origin {5 5}\
?  -leng 30
{"HW3690"}
```

SEE ALSO

remove_user_shape(2)
get_user_shapes(2)

create_via

Creates a new via

SYNTAX

```
status
create_via
   -at point
   [-name name]
   [-master master_name]
   [-auto]
   [-net net_name]
   [-no_net]
   [-route_type route_type]
   [-orient orient]
   [-type via_type]
   [-col num_cols]
   [-row num_rows]
   [-x_pitch xpitch]
   [-y_pitch ypitch]
```

ARGUMENTS

-at *point*

Specifies the center of the created via or via array.

-name *name*

Specifies the optional name of the created via or via array.

-master *master_name*

Specifies the master name of the via.

Note: if this is not specified the value is taken from the mw_via_master global variable.

Note: this must be specified if -auto is not supplied

-auto

Automatically create via based on the nearest wire intersection at the specified creation coordinate.

This automatically sets the -col, -row, -x_pitch, -y_pitch, -master and -net options.

The user can override the automatic vaules for -col, -row, -x_pitch, -y_pitch and -net using explicit values.

Note: This option precludes the use of -master option

-net *net_name*

Specifies the net name of the net to connect to the via

-no_net

Specifies that no net should be used.

Note: When not using the -auto option one of -net or -no_net must be specified.

-route_type *route_type*

The route type of the via. One of:

user_enter User Entered signal_route, signal_route_detail Detail Routing signal_route_global Global Routing pg_ring Power Ground Ring pg_strap Power Ground Strap pg_macro_io_pin_conn Power Ground Macro to IO Pin pg_std_cell_pin_conn Power Ground Std Cell to IO Pin clk_ring Clock Ring clk_strap Clock Strap clk_zero_skew_route Clock Zero Skew Route bus Bus shield, shield_fixed Fixed Shield shield_dynamic Dynamic Shield fill_track, clk_fill_track Fill Track

If this is not specified it defaults to signal_route.

-orient *orient*

Specifies the orientation of the via or via array using either DEF or Floorplan Compiler notation.

The following values are allowed:

N, W, S, E, FN, FS, FE, FW

NW, NE, EN, ES, SE, SW, WN, WS

0, 90, 180, 270, 0-mirror, 90-mirror, 180-mirror, 270-mirror

The following values are synominous:

N , NE, 0 W , WN, 90 S , SW, 180 E , ES, 270 FN, NW, 0-mirror FS, SE, 180-mirror FE, EN, 270-mirror FW, WS, 90-mirror

If this is not specified it defaults to 0.

-type *type*

Specifies which type of via is going to be created. One of:

via Via via_array Via Array

If no type has been specified, a via will be created unless one of the vai array specific options is used.

-col *num_cols*

Number of via array columns

-row *num_rows*

Number of via array rows

-x_pitch *xpitch*

X distance between cuts in the via array

-y_pitch *ypitch*

Y distance between cuts in the via array

DESCRIPTION

This command creates a new via or via_array

NOTES

Snapping is done automatically using global snap settings.

EXAMPLES

The following example create a via

```
> create_via -center {100 100} -route_type clock -master VIA12 -no_net
```

SEE ALSO

create_net_shape(2)
create_user_shape(2)
create_placement_blockage(2)
create_route_guide(2)
create_terminal(2)
create_text(2)

create_via_master

Creates a design via master using parameters or a geometry list.

SYNTAX

```
status create_via_master
   -name via_master_name
   -cut_layer_name cutLayerName
   -lower_layer_name loLayerName
   -upper_layer_name upLayerName
   [ -rectangles {{layerName {llx lly urx ury}} ... }]
   [ -cut_width cutWidth]
   [ -cut_height cutHeight]
   [ -lower_layer_enc_width loEncWidth]
   [ -lower_layer_enc_height loEncHeight]
   [ -upper_layer_enc_width upEncWidth]
   [ -upper_layer_enc_height upEncHeight]
   [ -min_cut_spacing minCutSpacing]
```

Data Types

```
   via_master_name  string
   cutLayerName     layer
   loLayerName      layer
   upLayerName      layer
   rectList         list
   cutWidth         float
   cutHeight        float
   loEncWidth       float
   loEncHeight      float
   upEncWidth       float
   upEncHeight      float
   minCutSpacing    float
```

ARGUMENTS

-name *via_master_name*

The name of the via master you want to create.

-cut_layer_name *cutLayerName*

The cut layer name of the via master.

-lower_layer_name ***loLayerName***

The lower metal layer name of the via master.

-upper_layer_name ***upLayerName***

The upper metal layer name of the via master.

-rectangles ***rectList***

The list of rectangles describing the geometry of the via master. If the **-rectangles** option is used, the parameter arguments listed below are ignored by the command.

-cut_width ***cutWidth***

The cut width of the via master, in microns.

-cut_height ***cutHeight***

The cut height of the via master, in microns.

-lower_layer_enc_width ***loEncWidth***

The lower metal layer enclosure width of the via master, in microns.

-lower_layer_enc_height ***loEncHeight***

The lower metal layer enclosure height of the via master, in microns.

-upper_layer_enc_width ***upEncWidth***

The upper metal layer enclosure width of the via master, in microns.

-upper_layer_enc_height ***upEncHeight***

The upper metal layer enclosure height of the via master, in microns.

-min_cut_spacing ***minCutSpacing***

The minimum spacing between cuts when the via master is used to form an array.

DESCRIPTION

This command can be used to create a new via master when the technology file does not already have a via master that matches the required geometry. The working cell in which you want to use the via must be open before you create the new via master.

You can create the new via master either by specifying a list of geometries using the **-rectangles** option or by specifying the width, height, lower-layer enclosure, upper-layer enclosure, and minimum cut spacing characteristics of the via, similar to a technology file definition.

To create a single-cut, symmetrical via master, use the height, width, enclosure, and minimum spacing options. To create a multi-cut via master such as an H-shape, use the **-rectangles** option.

When you save the current working cell, the via master is also saved along with the current cell. You can

use the new via master any number of times.

If you specify a geometry list using the **-rectangles** option and you also specify parameters such as width, height, enclosure, and minimum spacing, the parameter settings are ignored and a warning is issued.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following command creates a single-cut via master using parameters:

```
prompt> create_via_master \
 -name VIA1 \
 -cut_layer_name CUT \
 -lower_layer_name METAL1 \
 -upper_layer_name METAL2 \
 -cut_width 0.22 \
 -cut_height 0.22 \
 -lower_layer_enc_width 0.1 \
 -lower_layer_enc_height 0.1 \
 -upper_layer_enc_width 0.01 \
 -upper_layer_enc_height 0.01 \
 -min_cut_spacing 0.25
```

Using parameters always creates a symmetric, single-cut via master.

The following command creates an H-shaped via master by specifying a geometry list:

```
prompt> create_via_master \
 -name VIA1 \
 -cut_layer_name CUT \
 -lower_layer_name METAL1 \
 -upper_layer_name METAL2 \
 -rectangles { \
         {METAL1 {-0.130 -0.035 0.130 0.035}}   \
         {CUT {0.035 -0.035 0.100 0.035}}   \
         {CUT {-0.100 -0.035 -0.035 0.035}}   \
         {METAL2 {-0.100 -0.065 -0.035 0.065}}  \
         {METAL2 {0.035 -0.065 0.100 0.065}} \
       }
```

SEE ALSO

```
create_via(2)
get_via_masters(2)
```

create_voltage_area

Creates a voltage area at the specified region for providing placement constraints of cells associated with the region.

SYNTAX

```
collection_handle create_voltage_area
   [-color color]
   [-bounding_box rectangle | -poly poly]
   [-guardband point]
   [-priority priority]
   [-cells cell_list]
   name
```

Data Types

```
   color      string
   rectangle  list
   poly       list
   point      list
   priority   int
   name       string
   cell_list  list
```

ARGUMENTS

-color *color*

Specifies the color for the new voltage area display. The valid values for *color* are the following:

red
green
blue
magenta
cyan
yellow
orange
purple
brown
aqua
salmon

The default color is defined by the **mw_va_color** Tcl variable. You can view its value by using

printvar and can set a new value for it by using **set**. The default is to use **mw_va_color**.

-bounding_box ***rectangle***

Specifies a rectangular area in which to create the voltage area. The rectangle coordinates are relative to the current design. The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

The **-bounding_box** and **-poly** options are mutually exclusive.

-poly ***poly***

Specifies a polygonal area in which to create the voltage area. The polygon coordinates are relative to the current design. The format of a polygon specification is {{*x1 y1*} {*x2 y2*} ... {*xn yn*} {*x1 y1*}}, in which the *n* must be no lower than *4*.

The **-bounding_box** and **-poly** options are mutually exclusive.

-guardband ***point***

Specifies the *x* and *y* guard width. Guardband is the spacing along the boundary of the voltage area where cells cannot be placed because of the lack of power supply rails. The format of a point specification is {*x y*}.

The default guardband is defined by the **mw_va_guardband** Tcl variable. You can view its value with **printvar** and can set a new value for it by using **set**. The default is **mw_va_guardband**.

-priority ***priority***

Indicates the priority of the voltage area. When two voltage areas overlap, the voltage area with the higher assigned priority takes precedence.

The default *priority* is defined by the **mw_va_priority** Tcl variable. You can view its value by using **printvar** and can set a new value for it by using *set*. The default is **mw_va_priority**.

-cells ***cell_list***

Specifies the list of the cells to be associated with the newly created voltage area. Each element in *cell_list* is either a collection or a pattern matching cell names in the current design.

name

Specifies the name of the voltage area to be created.

DESCRIPTION

This command enables you to create voltage area at the specified region. A voltage area provides placement constraints of cells associated with the region.

EXAMPLES

The following example creates a voltage area.

```
prompt> printvar mw_va_color
mw_va_color          = "red"
prompt> printvar mw_va_guardband
mw_va_guardband      = "2 2"
prompt> printvar mw_va_priority
mw_va_priority       = "1"
prompt> set mw_va_color orange
orange
prompt> set mw_va_guardband {1 2}
1 2
prompt> create_voltage_area -priority 3 \
-poly {{200 195} {320 195} {320 205} {200 205} {200 195}} \
youreahero
{"youreahero"}
prompt> create_voltage_area -poly \
{{-350 100} {100 100} {100 130} {-350 130} {-350 100}} \
templeoftheking -color cyan -cells buffdaG1B2I1_1
{"templeoftheking"}
```

SEE ALSO

```
get_voltage_areas(2)
remove_voltage_area(2)
```

current_instance

Sets the working instance object and enables other commands to be used on a specific cell in the design hierarchy.

SYNTAX

```
string current_instance
   [instance]
```

Data Types

```
   instance     string
```

ARGUMENTS

instance

Specifies the working cell.

The **current_instance** command operates with a variety of *instance* arguments:

- If you do not specify the *instance* argument, the focus is returned to the top level of the hierarchy.
- If you specify ".", the current instance is returned and no change is made.
- If you specify "..", the current instance is moved up one level in the design hierarchy.
  You can also nest the ".." directive in complex instance specifications. For example, ../../MY_INST attempts to move the context up two levels of hierarchy, and then down one level to the MY_INST cell.
- If you specify a valid cell at the current level of the hierarchy, the current instance is moved down to that level of the design hierarchy.
- You can traverse multiple levels of hierarchy in a single call to the **current_instance** command by separating multiple cell names with slashes.
  For example, if you specify U1/U2, the current instance is moved down two levels of hierarchy if both cells exist at the current levels in the design hierarchy.

More complex examples of *instance* arguments are described below in EXAMPLES.

DESCRIPTION

The **current_instance** command sets the working instance. An instance is a cell embedded in the hierarchy of a design. Usually you define an instance to set or get attributes on a cell.

To display the instances available at the current level of a design hierarchy, use the **list_instances** command.

.prod syn

The **current_design** command changes the working design, setting the current instance to the top level of the new current design. .prod all

The **current_instance** command traverses the design hierarchy similar to the way the UNIX **cd** command traverses the file hierarchy.

Note that the **current_instance** command does not work on leaf cells. If you attempt to run the **current_instance** command on a leaf cell, an error occurs.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example uses the **current_instance** command to move up and down the design hierarchy and the **list_instances** command to show the available instances at each point in the hierarchy:

```
prompt> current_design .
Current design is 'TOP'.
{TOP}

prompt> list_instances
U1 (ADDER)      U2 (SUBTRACTOR)

prompt> current_instance U1
Current instance is 'TOP/U1'.
/TOP/U1

prompt> current_instance .
Current instance is 'TOP/U1'.
/TOP/U1

prompt> list_instances
U1 (FULL_ADDER) U2 (FULL_ADDER) U3 (FULL_ADDER) U4 (FULL_ADDER)

prompt> current_instance U3
Current instance is 'TOP/U1/U3'.
/TOP/U1/U3
```

```
prompt> current_instance "../U4"
Current instance is 'TOP/U1/U4'.
/TOP/U1/U4

prompt> current_instance
Current instance is the top-level of design 'TOP'.
```

.prod syn

In the following example, changing the current design resets the **current_instance** to the top level of the new design hierarchy:

```
prompt> current_design
Current design is 'TOP'.
{TOP}

prompt> current_instance "U2/U1"
Current instance is 'TOP/U2/U1'.
/TOP/U2/U1

prompt> current_design ADDER
Current design is 'ADDER'.
{ADDER}

prompt> current_instance .
Current instance is the top-level of design 'ADDER'.
```

.prod all

The following example uses **current_instance** to go to an instance of another design. The current design is set by the new design whose name is the name after the first slash of the given instance name.

```
prompt> current_design .
Current design is 'TOP'.
{TOP}

prompt> current_instance U1
Current instance is 'TOP/U1'.
/TOP/U1

prompt> current_instance /TOP/U2
Current instance is 'TOP/U2'.
/TOP/U2

prompt> current_instance /ALARM_BLOCK/U6
Current instance is 'ALARM_BLOCK/U6'.
/ALARM_BLOCK/U6
```

SEE ALSO

```
current_design(2)
list_designs(2)
list_instances(2)
```

current_mw_cel

Gets (or sets) the working Milkyway cel in the tool.

SYNTAX

```
collection current_mw_cel [mw_cel]
```

Data Types

mw_cel cel name

ARGUMENTS

mw_cel

Specifies a Milkyway cel to be set as the current Milkyway cel. By default, the command uses the current working Milkyway cel.

You can specify Milkyway cel by name, name pattern, or the Milkyway cel collection's name. For example, *top* matches a Milkyway cel named *top* in the current library and *top** matches all Milkyway cels with names beginning with *top*. This option must specify only one Milkyway cel. Multiple Milkyway cels cannot be set as the current Milkyway cel.

DESCRIPTION

This command gets (or sets) the working Milkyway cel for many other commands.

If you do not specify an *mw_cel* value and the current_mw_cel was previously set, the command returns the value of the current Milkyway cel.

If you specify an *mw_cel* value, the command first determines if the specified Milkyway cel is an open Milkyway cel. If the Milkyway cel is not open, it informs you to open the open the Milkyway cel first. If it is an open Milkyway cel, the command sets the current Milkyway cel from the specified Milkyway cel.

The following three commands change the value of the current Milkyway cel:

- The **open_mw_cel** command automatically sets the opened Milkyway cel as the current Milkyway cel.
- The **close_mw_cel** command clears the current Milkyway cel if the closed Milkyway cel is the current Milkyway cel.
- The **close_mw_lib** command clears the current Milkyway cel if the current Milkyway cel belongs to that closed library.

EXAMPLES

The following example uses the **current_mw_cel** command to show the current context and changes the context from one Milkyway cel to another:

```
prompt> current_mw_cel
{"top"}

prompt> current_mw_cel [get_mw_cel ADDER]
{"ADDER"}

prompt> current_mw_cel
{"ADDER"}
```

The following example uses the **close_mw_cel** command to close the working Milkyway cel from the tool:

```
prompt> current_mw_cel
{"TOP"}

prompt> close_mw_cel
1

prompt> current_mw_cel
Error: No mw cel is open.
```

SEE ALSO

close_mw_cel(2)
copy_mw_cel(2)
create_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
remove_mw_cel(2)
rename_mw_cel(2)
save_mw_cel(2)

current_mw_lib

Gets the current Milkyway library.

SYNTAX

```
collection current_mw_lib
```

DESCRIPTION

This command gets the current Milkyway library. The **current_mw_lib** command returns a collection of current Milkyway libraries if one exists.

Many Milkyway library-related commands use the current Milkyway library by default. The current Milkyway library is automatically set when the **open_mw_lib** command opens a Milkyway library to use.

EXAMPLES

The following example returns the current Milkyway library in the current session.

```
prompt> current_mw_lib
{"design"}
```

SEE ALSO

```
close_mw_lib(2)
copy_mw_lib(2)
create_mw_lib(2)
open_mw_lib(2)
rename_mw_lib(2)
report_mw_lib(2)
update_mw_lib(2)
```

decrypt_lib

Decrypts a mw library, whose technology info is locked.

SYNTAX

```
status decrypt_lib
   [-format {mwlib}]
   -key key_string
   lib_name
```

Data Types

```
   key_string          string
   lib_name            string
```

ARGUMENTS

lib_name

Specifies the name of the mw library to decrypt.

-format *{mwlib}*

Specifies the file format. By default, format is mwlib.

-key *key_string*

Specifies the key string to unlock mw library technology info, If wrong string supplied(that is different from the key string used to encrypt the mw library), the mw library can't be decrypted.

DESCRIPTION

```
This command can decrypts an encrypted mw library if the library
was encrypted by command "encrypt_lib -format mwlib -key keyStr".

If a library is encrypted natively by the creation from encrypted
```

tech file, it can't be decrypted by this command.

EXAMPLES

The following example shows the usage of this command: Decrypt a mw library "my_lib" with a key string.

```
prompt> decrypt_lib -key {ki08$#98fa?ipo} my_lib
```

SEE ALSO

encrypt_lib(2)

define_antenna_accumulation_mode

Defines an antenna accumulation mode route rule.

SYNTAX

```
status_value define_antenna_accumulation_mode
   [mw_lib | -lib lib_id]
   [-cut_to_metal]
   [-metal_to_cut]

mw_lib list


lib_id string
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be updated. The value of *mw_lib* can be a library name or a one-element collection of a library. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-lib *lib_id*

Specifies the ID of the Milkyway library to be updated. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-cut_to_metal

When *-cut_to_metal* is on, via ratios will be accumulated to metal ratios. Default is off.

-metal_to_cut

When *-cut_to_metal* is on, metal ratios will be accumulated to cut ratios. Default is off.

DESCRIPTION

This command defines an antenna accumulation mode route rule and stores it in the library. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> define_antenna_accumulation_mode \
  -cut_to_metal -metal_to_cut
```

SEE ALSO

report_antenna_rules(2)
remove_antenna_rules(2)

define_antenna_area_rule

Defines antenna area rule for the specified mode and stores it in the library. Customers would like to consider metal area as antenna rule as well as max antenna ratio, because when the metal area become too big, the electric charge will be pooled and the gate will be damaged.

The antenna area rule can be stored in either physical reference libraries or a design library. The word "library" in this man page can refer to either of them by usage context in real case.

SYNTAX

```
status define_antenna_area_rule
   [-library mw_lib | -lib_id lib_id]
   -mode <ignore_lower_layers|include_lower_layers|include_all_lower_layers>
   -max_area <area>
   [-diode_distance <distance>]

mw_lib string


lib_id string


mode string


max_area float


diode_distance float
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to which the antenna area rule is to be stored. The value of *mw_lib* can be a library name or a one-element collection of a library. By default, the command uses the current Milkyway library. This argument and **-lib_id** options are mutually exclusive.

-lib_id *lib_id*

Specifies the ID of the Milkyway library, to which antenna area rule is to be stored. By default, the command uses the current Milkyway library. The argument and **-library** options are mutually exclusive.

-mode *mode*

Defines the way metal(routing) areas are computed. Valid values: ignore_lower_layers-Calculate metal area, ignoring all lower-layer segments. include_lower_layers-Calculate metal area, including lower-layer segments to the input pins. include_all_lower_layers-Calculate metal area, including all lower-layer segments.

For example. a1, a2, a3, a4, a5 stand for wire area that all connected to gate, we omit the via connection wire area in this case.

__a3__ --metal3 wire area __a2__| |__a4__ --metal2 wire area [gate]__a1__| |__a5__ --metal1 wire area

For different mode, when calculate total routing area for metal3 layer: ignore_lower_layers: total_routing_area = a3; include_lower_layers: total_routing_area = a1+a2+a3; include_all_lower_layers: total_routing_area = a1+a2+a3+a4+a5;

Note that only one rule can be defined for every mode. If two commands contain the same mode number, the second command will overwrite the first one.

-max_area *area*

This is maximum allowable metal area for a gate, by square distance unit of current edit library. The metal area (connected to the gate) calculated base on mode can't exceed this constraint.

If this value is zero, the antenna area rule will be ignored.

Valid value: any non-negative number.

-diode_distance *distance*

This is maximum allowable point to point distance from inserted diode to gate, by distance unit of current edit library. It's used to prevent diode cell being inserted far away from gate, so as to guarantee the efficiency of antenna fixing by diode insertion.

If this value is zero, the antenna fixing won't use diode insertion but by wire re-routing.

Valid value: any non-negative number.

DESCRIPTION

This command defines an antenna area rule for the specified mode and stores it in the library. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> define_antenna_area_rule -mode ignore_lower_layers -max_area 500 -diode_distance 50.5
```

SEE ALSO

```
define_antenna_rule(2)
define_antenna_layer_ratio_scale(2)
define_antenna_layer_rule(2)
report_antenna_rules(2)
remove_antenna_rules(2)
```

define_antenna_layer_ratio_scale

Creates an antenna layer ratio route rule.

SYNTAX

```
status_value define_antenna_layer_ratio_scale
   [mw_lib | -lib lib_id]
   -layer <layer_name>
   -layer_scale <layer_scale>
   -accumulate_scale <accumulate_scale>

mw_lib list


lib_id string


layer_name string


layer_scale float


accumulate_scale float
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be updated. The value of *mw_lib* can be a library name or a one-element collection of a library. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-lib *lib_id*

Specifies the ID of the Milkyway library to be updated. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-layer *layer_name*

Name of the layer.

-layer_scale *layer_scale*

Layer scale factor.

-accumulate_scale ***accumulate_scale***

Accumulation scale factor.

DESCRIPTION

This command creates an antenna layer ratio route rule. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> define_antenna_layer_ratio_scale -layer metal1 \
  -layer_scale 1000.00 -accumulate_scale 2.00
```

SEE ALSO

define_antenna_rule(2)
define_antenna_layer_rule(2)
report_antenna_rules(2)
remove_antenna_rules(2)

define_antenna_layer_rule

Defines an advanced antenna rule for the specified layer and stores it in the library.

SYNTAX

```
status_value define_antenna_layer_rule
   [mw_lib | -lib lib_id]
   -mode <mode>
   -layer <layer_name>
   -ratio <ratio> | -pratio <ratio> -nratio <ratio>
   -diode_ratio <diode_ratio>

mw_lib list


lib_id string


mode int


layer_name string


ratio float


pratio float


nratio float


diode_ratio list
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be updated. The value of *mw_lib* can be a library name or a one-element collection of a library. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-lib ***lib_id***

Specifies the ID of the Milkyway library to be updated. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-mode ***mode***

Defines the way antenna areas are computed. Valid values: 1-Uses polygon area, ignoring all lower-layer segments. 2-Uses polygon area, including all lower-layer segments to the input pins. 3-Uses polygon area, including all lower-layer segments. 4-Uses side-wall area, ignoring all lower-layer segments. 5-Uses side-wall area, including all lower-layer segments to the input pins. 6-Uses side-wall area, including all lower-layer segments.

Note that only one rule can be defined for every mode. If two commands contain the same mode number, the second command will overwrite the first one.

-layer ***layer_name***

Name of the valid metal layer or cut layer in the library.

-ratio ***ratio***

The maximum allowable ratio of the antenna area to the gate area if the antenna is not protected by any diode.

-pratio ***ratio***

The maximum allowable ratio of the antenna area to the pgate area if the antenna is not protected by any diode. This option must use together with -nratio option, and they can't be used with antenna mode = 2|5.

-nratio ***ratio***

The maximum allowable ratio of the antenna area to the ngate area if the antenna is not protected by any diode. This option must use together with -pratio option, and they can't be used with antenna mode = 2|5.

Valid values: Any number.

-diode_ratio ***diode_ratio***

Specify the allowable ratio in the antenna area to the gate area if the antenna is protected by a diode. Valid values: {v0 v1 v2 v3 [v4]}

If the output pin protection value is **dp**, the allowable ratio will be: ((dp + v1) * v2 + v3), if (dp) > (v0) layer_max_ratio, if (dp) <= (v0)

The **dp** is specified in the CLF file.

The default value of **diode_ratio** is **{0 0 1 0}**.

DESCRIPTION

This command defines an advanced antenna rule for the specified layer and stores it in the library. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> define_antenna_layer_rule -mode 4 -layer metal1 -ratio 400 -diode_ratio {0.336 -0.5 400 240
```

SEE ALSO

define_antenna_rule(2)
report_antenna_rules(2)
remove_antenna_rules(2)

define_antenna_rule

Defines an advanced antenna rule for the specified mode and stores it in the library.

SYNTAX

```
status_value define_antenna_rule
   [mw_lib | -lib lib_id]
   -mode <mode>
   -diode_mode <diode_mode>
   [-metal_ratio <metal_ratio>]
   [-cut_ratio <cut_ratio>]
   [-metal_pratio <metal_pratio>]
   [-metal_nratio <metal_nratio>]
   [-cut_pratio <cut_pratio>]
   [-cut_nratio <cut_nratio>]
   [-protected_metal_scale <metal_scale>]
   [-protected_cut_scale <cut_scale>]

mw_lib list


lib_id string


mode int


diode_mode int


metal_ratio float


cut_ratio float


metal_pratio float


metal_nratio float


cut_pratio float


cut_nratio float


metal_scale float
```

cut_scale float

ARGUMENTS

mw_lib

Specifies the Milkyway library to be updated. The value of *mw_lib* can be a library name or a one-element collection of a library. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-lib ***lib_id***

Specifies the ID of the Milkyway library to be updated. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-mode ***mode***

Defines the way antenna areas are computed. Valid values: 1-Uses polygon area, ignoring all lower-layer segments. 2-Uses polygon area, including all lower-layer segments to the input pins. 3-Uses polygon area, including all lower-layer segments. 4-Uses side-wall area, ignoring all lower-layer segments. 5-Uses side-wall area, including all lower-layer segments to the input pins. 6-Uses side-wall area, including all lower-layer segments.

Note that only one rule can be defined for every mode. If two commands contain the same mode number, the second command will overwrite the first one.

-diode_mode ***diode_mode***

Defines the protection capability of the diode. By default, all output pins are considered as a diode. Valid values: 0-Output pin cannot protect antenna. 1-Any diode can provide unlimited protection. 2-Diode protection is limited; if more than one diode is connected, the largest value of max-antenna-ratio of all diodes will be used. 3-Diode protection is limited; if more than one diode is connected, the sum of max-antenna-ratio of all diodes will be used. 4-Diode protection is limited; if more than one diode is connected, the sum of all diode-protection value of all diodes will be used to compute max-antenna-ratio. 5-Diode protection is limited; the maximum diode-protection value of all diodes will be used to calculate the equivalent gate area. 6-Diode protection is limited; the sum of all diode-protection value of all diodes will be used to calculate the equivalent gate area. 7-Diode protection is limited; the maximum diode-protection value of all diodes will be used to calculate the equivalent metal area. 8-Diode protection is limited; the sum of all diode-protection value of all diodes will be used to calculate the equivalent metal area.

-metal_ratio ***metal_ratio***

Maximum allowable ratio for metal area to gate size if the metal layer is not defined with **define_antenna_layer_rule**.

If this value is zero, the ratio will be ignored.

Valid value: any non-negative number.

-cut_ratio ***cut_ratio***

Maximum allowable ratio for cut area to gate size if the cut layer is not defined with **define_antenna_layer_rule**.

If this value is zero, the ratio will be ignored.

Valid value: any non-negative number.

-metal_pratio *metal_pratio*

Maximum allowable ratio for antenna area (metal) to p-gate area.

-metal_nratio *metal_nratio*

Maximum allowable ratio for antenna area (metal) to n-gate area.

-cut_pratio *cut_pratio*

Maximum allowable ratio for antenna area (cut) to p-gate area.

-cut_nratio *cut_nratio*

Maximum allowable ratio for antenna area (cut) to n-gate area.

-protected_metal_scale *metal_scale*

The option is used when mode is 2 or 5 only. The area of the metal layer that is protected by diode will be scaled by this value. By default, the value is set to 1.0.

If this value is zero, the scale will be ignored.

Valid value: any non-negative number.

-protected_cut_scale *cut_scale*

The option is used when mode is 2 or 5 only. The area of the cut layer that is protected by diode will be scaled by this value. By default, the value is set to 1.0.

If this value is zero, the scale will be ignored.

Valid value: any non-negative number.

DESCRIPTION

This command defines an advanced antenna rule for the specified mode and stores it in the library. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> define_antenna_rule -mode 1 -diode_mode 2 \
```

```
        -metal_ratio 1000 -cut_ratio 0
```

SEE ALSO

define_antenna_layer_ratio_scale(2)
define_antenna_layer_rule(2)
report_antenna_rules(2)
remove_antenna_rules(2)

define_user_attribute

Defines a new user-defined attribute.

SYNTAX

```
int define_user_attribute
   -type data_type
   -class class_list
   [-range_min min]
   [-range_max max]
   [-one_of values]
   [-quiet]
   attr_name

data_type string
class_list list
min double
max double
values list
attr_name string
```

ARGUMENTS

-type *data_type*

Specifies the data type of the attribute. The supported data types are **string**, **int**, **float**, **double**, and **Boolean**.

-class *class_list*

Specifies the list of class names for the user-defined *attr_name* attribute. The following is a list of valid values for the elements of *class_list*:

design
port
cell
pin
net
lib
lib_cell
lib_pin
clock
bound

```
net_shape
placement_blockage
route_guide
route_shape
terminal
text
```

-range_min ***min***

Specifies the minimum value for numeric ranges. This option is valid only when the data_type is **int** or **double**. Specifying a minimum constraint without a maximum constraint creates an attribute that accepts a value greater than or equal to *min*.

-range_max ***max***

Specifies maximum value for numeric ranges. This option is valid only when the data_type is **int** or **double**. Specifying a maximum constraint without a minimum constraint creates an attribute that accepts a value less than or equal to *max*.

-one_of ***values***

Provides a list of allowable strings. This option is valid only when the data type is *string.*

-quiet

Turns off the warning message that would otherwise be issued if the attribute or classes are improper.

attr_name

Specifies the name of the attribute.

DESCRIPTION

This command defines a new attribute. You can use the **list_attributes** command to list the attributes you defined. The return value is set to **1** if the operation is successful; otherwise, it is set to **0**.

Note that the definition for a user-defined attribute can be persistent if it has been operated on to a specified object and stored into the database.

EXAMPLES

The following example defines *attr_1* as greater than or equal to **2** and less than or equal to **3.2** on classes **cell** and **net**.

```
prompt> define_user_attribute -class {cell net} -type double \
-range_max 3.2 -range_min 2 attr_1
Info:User-defined attribute 'attr_1' on class 'cell'.
Info:User-defined attribute 'attr_1' on class 'net'.
1
```

The following example defines *attr_2* to **string** with a value of either **true** or **false** on classes **cell** and **net**..

```
prompt> define_user_attribute -class net -type string \
-one_of {true false} attr_2
Info:User-defined attribute 'attr_2' on class 'net'.
1
```

The following example shows how you can list the attribute definitions by using the **list_attribute** command.

```
prompt> list_attributes
****************************************
Report : List of Attribute Definitions
Design :
Version:
Date   : Mon Dec 22 17:08:51 2003
****************************************

Properties:
    A - Application-defined
    U - User-defined
    I - Importable from db (for user-defined)

Attribute Name   Object   Type     Properties  Constraints
-------------------------------------------------------
attr_1           cell     double   U           2 to 3.2
attr_1           net      double   U           2 to 3.2
attr_2           net      string   U           true, false
```

SEE ALSO

```
get_attribute(2)
list_attributes(2)
remove_attribute(2)
set_attribute(2)
```

detach_file

Detaches a file from a library or design.

SYNTAX

```
int detach_file
   [-class class_name]
   [-on object_spec]
   [-to copy_location]
   [-preserve]
   name

class_name string
object_spec list
copy_location string
name string
```

ARGUMENTS

-class *class_name*

Specifies the class name of the object on which the attached file is located.

-on *object_spec*

Specifies a library or design object from which the file is to be detached. The value of *object_spec* can be either a collection of exactly one object or the name pattern whose class is specified by **-class** option. The specified object should be opened in write mode.

By default, the object is just the current design, if available; otherwise, it is determined by the opened main library.

-to *copy_location*

Copies the attached file to the specifies location if the option is provided.

-preserve

Preserves the attached file in the directory of MW. Only detaches the file from the object.

name

Specifies a pseudonym to be associated with the attached file.

DESCRIPTION

This command detaches a file from a library or design object. The return value is set to **1** if the operation is successful; otherwise, it is set to **0**.

EXAMPLES

This example first attaches a file named *new_attach1* with a pseudonym name *attach2* provided to the library named *design* and then detaches the file from *design*.

```
prompt> attach_file -pseudonym new_attach1 -to [get_libs design] attach2
1
prompt> detach_file -from [get_libs design] new_attach1
1
```

SEE ALSO

attach_file(2)

disconnect_net

Disconnects a net from pins or ports.

SYNTAX

```
int disconnect_net
   net
   object_list
   -all

net          list
object_list  list
```

ARGUMENTS

net

Specifies the net to be disconnected. It can be a name or collection. A net must exist in the current design.

object_list

Specifies the pins and ports disconnected from the net. Only pins and ports existing in the current design are specified. Either *object_list* or **-all** must be specified.

-all

Specifies to break all connections on the net.

DESCRIPTION

The **disconnect_net** command breaks the connections between a net in the current design and its pins or ports. The net, pins, and ports are not removed.

This command accepts only scalar (single bit) nets, but not bused nets.

To connect nets, use the **connect_net** command. To display the pins and ports connected to a net, use the **all_connected** command.

EXAMPLES

The following examples show nets being disconnected using the **disconnect_net** command.

```
prompt> disconnect_net NET0 [get_ports A1]
Disconnecting net 'NET0' from port 'A1'.

prompt> disconnect_net NET0 [get_pins U1/A]
Disconnecting net 'NET0' from pin 'U1/A'.
```

The following example shows all connections on a net being broken using **disconnect_net**.

```
prompt> disconnect_net MY_NET_1 -all
Disconnecting net 'MY_NET_1' from port 'PORT1'.

prompt> all_connected [get_nets MY_NET_1]
{}
```

SEE ALSO

all_connected(2)
connect_net(2)
create_net(2)
current_design(2)
get_nets(2)
remove_net(2)

dump_tlu_plus_file

Dumps TLU+ information for a Milkyway library.

SYNTAX

```
status_value dump_tlu_plus_file
   [-library mw_libs]
   [-lib_id lib_id]
   -output tlu_plus_file
```

Data Types

```
   mw_libs        string
   lib_id         string
   tlu_plus_file  string
```

ARGUMENTS

-library *mw_libs*

Specifies the Milkyway libraries for which to dump TLU+ information. The *mw_libs* value can be one or more library names or a list of a collection of libraries. By default, the command uses the current Milkyway library. The **-library** and **-lib_id** options are mutually exclusive.

-lib_id *lib_id*

Specifies the ID of the Milkyway library for which to dump TLU+ information. By default, the command uses the current Milkyway library. The **-library** and **-lib_id** options are mutually exclusive.

-output *tlu_plus_file*

Specifies the output TLU+ file name.

DESCRIPTION

This command dumps TLU+ information for a Milkyway library. It returns a status indicating success or failure.

EXAMPLES

The following example dumps the TLU+ information of the current Milkyway library to a file named *my_tlu_plus_file*.

```
prompt> dump_tlu_plus_file -output my_tlu_plus_file
1
```

SEE ALSO

`replace_tlu_plus_file(2)`

enable_ipv6

Enable ipv6 on value more than 0 or ipv4 is enabled for CDPL ipv4 is on by default

SYNTAX

```
status enable_ipv6
   value
```

Data Types

```
   value int
```

ARGUMENTS

value

Specifies the IP version is IPv4 or IPv6. IPv4 is default or turned on when the value is not more than 0 otherwise IPv6 is turned on when the value is more than 0.

DESCRIPTION

This command defines CDPL works on IPv4 or IPv6 mode.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

```
enable_ipv6 1
1
```

encrypt_lib

Encrypts a VHDL/TF source library file, or a mw library.

SYNTAX

```
status encrypt_lib
   [-format {vhdl | tf | mwlib}]
   [-output encrypted_file]
   [-key key_string]
   file_name
```

Data Types

```
   encrypted_file      string
   key_string          string
   file_name           string
```

ARGUMENTS

file_name

Specifies the name of the VHDL file/TF file/MW library to encrypt.

-format *{vhdl | tf |mwlib}*

Specifies the source file format that will be encrypted. By default, format is vhdl or mwlib, depends on the file specified is a mw library or a normal file. Since different format using different algorithm in encryption, it's very important to choose right format.

-output *encrypted_file*

Specifies the output filename to which the encrypted output VHDL/TF file is to be written. It's unnecessary to specify this option when encrypt a mw library (that's format = mwlib). By default, the output file is named file_name.E and is written to the current directory.

NOTE: If you specify an output filename that already exists, the command overwrites the file without warning.

-key *key_string*

Specifies the key string to lock mw library technology info, it's required when encrypt a mw library,

while unnecessary when encrypt VHDL/TF file.

DESCRIPTION

```
This command encrypts a VHDL/TF source library file, or mw library
to protect ASIC vendors' sensitive technology data.
The encrypted vhdl file is read and simulated by vhdlan and vhdlsim,
respectively, in the same way as a nonencrypted file.

The encrypted tf file can be taken by "create_mw_lib".

A library created from a encrypted tech file is called "encrypted library",
which prevents users dumping/reporting/replacing technology info from/for it.
Users can also use this command "encrypt_lib -format mwlib -key $keystr $mwlib"
to encrypt an existing mw library as "encrypted library".
The difference of these two kinds of "encrypted library" is: The encrypted
library generated thru encrypted tech file can not be decrypted by command
"decrypt_lib", while the other can when right key string supplied.
```

EXAMPLES

The following examples show the usage of this command:

1. Encrypt the technology library *my_file.vhd*. The output file is named *my_file.E* and is written to the current directory.

```
prompt> encrypt_lib my_file.vhd
```

2. Encrypt the technology file *my_tech.tf*, the output file is named

tech.Enc and is writtern to the current directory.

```
prompt> encrypt_lib -format tf -output tech.Enc my_tech.tf
```

3. Encrypt a mw library "my_lib" with a key string.

```
prompt> encrypt_lib -format mwlib -key {ki08$#98fa?ipo} my_lib
```

SEE ALSO

```
decrypt_lib(2)
```

extend_mw_layers

Extends Milkyway database layer number support to 4095 layers, using layers 4001 through 4095 as the system-reserved layers.

SYNTAX

```
status extend_mw_layers
```

DESCRIPTION

By default, the Milkyway database supports the usage of up to 255 layers, numbered 1 through 255. Layers 1 thorough 187 are available as user-defined layers and routing layers, while layers 188 through 255 are reserved for the Milkyway system layers.

Milkyway system layers are the layers used to define blockages, route guides, and other physical features that guide the router. The Milkyway system layer numbers are fixed and cannot be assigned or changed by the user. User-defined layers are layers defined in the technology (.tf) file, such as metal layers and text layers.

Some advanced process and routing technologies require more than 255 layers. In these cases, you can extend the number of layers supported in a Milkyway library by executing the **extend_mw_layers** command. In that case, layers 1 thorough 4000 are available as user-defined layers, while layers 4001 through 4095 are reserved for the Milkyway system layers.

In both the default and extended layer modes, routing layers are restricted to layer numbers 1 through 187. Routing layers are the standard layers used by the IC Compiler router to make interconnections, including wire routes, vias, and contacts.

To create a new Milkyway library using extended layers, you must execute the **extend_mw_layers** command before the **create_mw_lib** command. Once created in extended layer mode, the Milkyway library cannot be changed back to use the default layer mode.

In future sessions, the tool automatically recognizes the layer mode of the Milkyway library and uses the correct Milkyway system layer numbers, from 188 to 255 in the default mode or from 4001 to 4095 in the extended layer mode.

EXAMPLES

The following example shows the usage of the command to extend layer number support to 4095 layers and to use layers 4001 through 4095 as the system-reserved layers in a new Milkyway library.

```
prompt> extend_mw_layers
1
prompt> create_mw_lib -technology mytech.tf my_lib_2
Start to load technology file mytech.tf
 ...
```

SEE ALSO

```
create_mw_lib(2)
set_stream_layer_map_file(2)
```

filter_collection

Filters an existing collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection filter_collection
   base_collection
   expression
   [-regexp]
   [-nocase]

base_collection   collection
expression        string
```

ARGUMENTS

base_collection

Specifies the base collection to be filtered. This collection is copied to the result collection. Objects are removed from the result collection if they are evaluated as **false** by the conditional *expression* value. Substitute the collection you want for *base_collection*.

expression

Specifies an expression with which to filter *base_collection*. Substitute the string you want for *expression*.

-regexp

Specifies that the =~ and !~ filter operators will use real regular expressions. By default, the =~ and !~ filter operators use simple wildcard pattern matching with the * and ? wildcards.

-nocase

Makes the pattern match case-insensitive. When you specify this option, you must also specify the **-regexp** option.

DESCRIPTION

Filters an existing collection, resulting in a new collection. The base collection remains unchanged. In many cases, commands that create collections support a **-filter** option that filters as part of the collection process, rather than after the collection has been made. This type of filtering is almost always more efficient than using the **filter_collection** command after a collection has been formed. The **filter_collection** command is most useful when you plan to filter the same large collection many times using different criteria.

The **filter_collection** command results in either a new collection or an empty string. A resulting new collection contains the subset of the objects in the input *base_collection*. A resulting empty string (the empty collection) indicates that the *expression* filtered out all elements of the input *base_collection*.

The basic form of the conditional expression is a series of relations joined together with AND and OR operators. Parentheses () are also supported. The basic relation contrasts an attribute name with a value through a relational operator. For example:

```
is_hierarchical == true and area <= 6
```

The relational operators are

```
==   Equal
!=   Not equal
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
=~   Matches pattern
!~   Does not match pattern
```

The basic relational rules are

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only **true** or **false**.

Existence relations determine if an attribute is defined or not defined for the object. For example:

```
sense == setup_clk_rise and defined(sdf_cond)
```

The existence operators are:

```
defined
undefined
```

These operators apply to any attribute as long as it is valid for the object class.

This command matches a regular expression matching in the same way as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the filter *expression*. Using rigid quoting with curly braces around regular expressions is considered best practice. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the

beginning or end of the expressions as needed. You can make the regular expression search case-insensitive by using the **-nocase** option.

EXAMPLES

The following example creates a collection of only hierarchical cells.

```
prompt> set a [filter_collection [get_cells *] \
"is_hierarchical == true"]
{"Adder1", "Adder2"}
```

The following example creates a collection of all nonmoded cell timing_arc objects in the current design.

```
prompt> set b [filter_collection \
[get_timing_arcs -of_objects [get_cells *]] \
"undefined(mode)"]
```

SEE ALSO

collections(2)
regexp(2)

flatten_cell

Flatten a Milkyway design that has a hierarchical reference to child cells to a specified hierarchy level.

SYNTAX

```
status_value flatten_cell
    -library lib_name
    [-cell cell_name]
    [-level level_number]
    [-view view_name]
    [-output_to_smash_view]
    [-cell_type {standard macro pad other}]
    [-keep_route_info]
    [-dont_change_original_view_port_name]
    [-port_name_mapping_file file_name]
    [-keep_text]
```

Data Types

```
lib_name  string
cell_name  string
level_number int
view_name  string
file_name  string
```

ARGUMENTS

-library

Specifies the name of the library that contains the cell(s) to be flattened.

-cell

Specifies the cell to be flattened. Use the * wildcard or leave this field blank to flatten all the cells.

-level

Specifies the hierarchy level for the cells to be flattened. Default value is 20.

-view

Determines the view to use to flatten the child cells. By default, the option is set to the 'CEL' View

which is the built-in view of a child cell in the hierarchy. You can specify another view, for example a .CONN view. If the specified view doesn't exist for some child cell(s), the CEL view will be used.

-output_to_smash_view

Specifies the view of the output. If it's off, the flattening result is written out in the original view. If it's on, the flattening result is output to SMASH view. Default it's off.

-cell_type

Specifies the list of cell types to be flattened. It can be a combination of **standard**, **macro**, **pad**, and **other**. Default is {standard macro pad}.

-keep_route_info

Keeps the detail route information, such as child cell hierarchy, route type, wire length, wire width, and data type. Default is off.

-dont_change_original_view_port_name

Keeps the original view port name, if the option is switched on. By default, this option is off, so the original port name is changed based on the rule defined in port_name_mapping_file.

-port_name_mapping_file

Specifies the port name mapping file in the following format, when the **-dont_change_original_view_port_name** option is not used:

cellName oldName newName

If you don't specify this file, the port name will not be changes during the flattening process.

-keep_text

Retains the text at all levels of hierarchy that are flattened. Default is off.

DESCRIPTION

Flattening a cell removes specified levels of hierarchy to make those hierarchical objects visible to top level in an import macro cell (Placed & Routed cell) or library cells (generated from GDS). The command is generally used before creating the FRAM view, or before blockage, pin and via extraction, or before preparing the reference library cells.

Flattening is used more often with macro abstracting than with abstracting standard cells.

Flattening an unopened cell changes the copy of the cells on the disk. Changing a cell that is opened changes the copy in the virtual memory. To retain the flattening of an open cell, save the cell before you close it.

EXAMPLES

The following example flattens the cells of type "macro" and "other" in the 'test' library, in the CEL view, with one level hierarchy, outputs the flattened cells to the view named SMASH, and does not change the port name of original view.

```
prompt> flatten_cell -library test
                     -view CEL
                     -output_to_smash_view
                     -level 1  F
                     -cell_type {macro other}
                     -dont_change_original_view_port_name
```

SEE ALSO

create_macro_fram(2)
close_mw_cel(2)
current_mw_cel(2)
open_mw_cel(2)
save_mw_cel(2)

force_exit

Close all open libraries without updates and exit the tool.

SYNTAX

```
force_exit
```

ARGUMENTS

None.

DESCRIPTION

The **force_exit** command causes the tool to exit, without asking if you want to write to the database.

EXAMPLES

The following example closes all libraries without updates and exits the tool.

```
prompt> force_exit
```

foreach_in_collection

Iterates over the elements of a collection.

SYNTAX

```
string foreach_in_collection
   itr_var
   collections
   body

itr_var     string
collections list
body        string
```

ARGUMENTS

itr_var

Specifies the name of the iterator variable.

collections

Specifies a list of collections over which to iterate.

body

Specifies a script to execute per iteration.

DESCRIPTION

The **foreach_in_collection** command is used to iterate over each element in a collection. You cannot use the **foreach** Tcl command to iterate over collections, because **foreach** requires a list, and a collection is not a list. Also, using **foreach** on a collection causes the collection to be deleted.

The arguments to **foreach_in_collection** parallel those of **foreach**: an iterator variable, the collections over which to iterate, and the script to apply at each iteration. All arguments are required. The

foreach_in_collection command does not allow a list of iterator variables.

During each iteration, *itr_var* is set to a collection of exactly one object. Any command that accepts *collections* as an argument accepts *itr_var*, because they are of the same data type (collection).

You can nest the **foreach_in_collection** command within other control structures, including **foreach_in_collection**.

If the body of the iteration is modifying the netlist, it is possible that all or part of the collection involved in the iteration will be deleted. The **foreach_in_collection** command is safe for such operations. If a command in the body causes the collection to be removed, at the next iteration, the iteration will end with a message indicating that the iteration ended prematurely.

An alternative to collection iteration is to use complex filtering to create a collection that includes only the desired elements, then apply one or more commands to that collection. If the order of operations does not matter, the following are equivalent. The first is an example without iterators.

```
set s [get_cells {U1/*}]
command1 $s
command2 $s
unset s
```

The following is the same example using **foreach_in_collection**.

```
foreach_in_collection itr [get_cells {U1/*}] {
  command1 $itr
  command2 $itr
}
```

For collections with large numbers of objects, the non-iterator version is more efficient, although both produce the same results if the commands are order-independent.

EXAMPLES

The following example removes the wire load model from all hierarchical cells in the current instance.

```
prompt> foreach_in_collection itr [get_cells *] {
?           if {[get_attribute $itr is_hierarchical] == "true"} {
?              remove_wire_load_model $itr
?            }
?          }
Removing wire load model from cell 'i1'.
Removing wire load model from cell 'i2'.
```

SEE ALSO

```
collections(2)
filter_collection(2)
query_objects(2)
```

format

'\" Note: do not modify the .SH NAME line immediately below!

format \- Format a string in the style of sprintf

SYNOPSIS

format ***formatString*** ?***arg arg ...***?

ARGUMENTS

none

INTRODUCTION

This command generates a formatted string in the same way as the ANSI C **sprintf** procedure (it uses **sprintf** in its implementation). *FormatString* indicates how to format the result, using **%** conversion specifiers as in **sprintf**, and the additional arguments, if any, provide values to be substituted into the result. The return value from **format** is the formatted string.

DETAILS ON FORMATTING

The command operates by scanning *formatString* from left to right. Each character from the format string is appended to the result string unless it is a percent sign. If the character is a **%** then it is not copied to the result string. Instead, the characters following the **%** character are treated as a conversion specifier. The conversion specifier controls the conversion of the next successive *arg* to a particular format and the result is appended to the result string in place of the conversion specifier. If there are multiple conversion

specifiers in the format string, then each one controls the conversion of one additional *arg*. The **format** command must be given enough *arg*s to meet the needs of all of the conversion specifiers in *formatString*.

Each conversion specifier may contain up to six different parts: an XPG3 position specifier, a set of flags, a minimum field width, a precision, a length modifier, and a conversion character. Any of these fields may be omitted except for the conversion character. The fields that are present must appear in the order given above. The paragraphs below discuss each of these fields in turn.

If the **%** is followed by a decimal number and a **$**, as in ``**%2$d**'', then the value to convert is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first *arg*. If the conversion specifier requires multiple arguments because of * characters in the specifier then successive arguments are used, starting with the argument given by the number. This follows the XPG3 conventions for positional specifiers. If there are any positional specifiers in *formatString* then all of the specifiers must be positional.

The second portion of a conversion specifier may contain any of the following flag characters, in any order: **\-** Specifies that the converted argument should be left-justified in its field (numbers are normally right-justified with leading spaces if needed). **+** Specifies that a number should always be printed with a sign, even if positive. *space* Specifies that a space should be added to the beginning of the number if the first character isn't a sign. **0** Specifies that the number should be padded on the left with zeroes instead of spaces. **#** Requests an alternate output form. For **o** and **O** conversions it guarantees that the first digit is always **0**. For **x** or **X** conversions, **0x** or **0X** (respectively) will be added to the beginning of the result unless it is zero. For all floating-point conversions (**e**, **E**, **f**, **g**, and **G**) it guarantees that the result always has a decimal point. For **g** and **G** conversions it specifies that trailing zeroes should not be removed.

The third portion of a conversion specifier is a number giving a minimum field width for this conversion. It is typically used to make columns line up in tabular printouts. If the converted argument contains fewer characters than the minimum field width then it will be padded so that it is as wide as the minimum field width. Padding normally occurs by adding extra spaces on the left of the converted argument, but the **0** and **\-** flags may be used to specify padding with zeroes on the left or with spaces on the right, respectively. If the minimum field width is specified as * rather than a number, then the next argument to the **format** command determines the minimum field width; it must be a numeric string.

The fourth portion of a conversion specifier is a precision, which consists of a period followed by a number. The number is used in different ways for different conversions. For **e**, **E**, and **f** conversions it specifies the number of digits to appear to the right of the decimal point. For **g** and **G** conversions it specifies the total number of digits to appear, including those on both sides of the decimal point (however, trailing zeroes after the decimal point will still be omitted unless the **#** flag has been specified). For integer conversions, it specifies a minimum number of digits to print (leading zeroes will be added if necessary). For **s** conversions it specifies the maximum number of characters to be printed; if the string is longer than this then the trailing characters will be dropped. If the precision is specified with * rather than a number then the next argument to the **format** command determines the precision; it must be a numeric string.

The fifth part of a conversion specifier is a length modifier, which must be **h** or **l**. If it is **h** it specifies that the numeric value should be truncated to a 16-bit value before converting. This option is rarely useful. The **l** modifier is ignored.

The last thing in a conversion specifier is an alphabetic character that determines what kind of conversion to perform. The following conversion characters are currently supported: **d** Convert integer to signed decimal string. **u** Convert integer to unsigned decimal string. **i** Convert integer to signed decimal string; the integer may either be in decimal, in octal (with a leading **0**) or in hexadecimal (with a leading **0x**). **o** Convert integer to unsigned octal string. **x** or **X** Convert integer to unsigned hexadecimal string, using digits ``0123456789abcdef'' for **x** and ``0123456789ABCDEF'' for **X**). **c** Convert integer to the Unicode

character it represents. **s** No conversion; just insert string. **f** Convert floating-point number to signed decimal string of the form *xx.yyy*, where the number of *y*'s is determined by the precision (default: 6). If the precision is 0 then no decimal point is output. **e** or **e** Convert floating-point number to scientific notation in the form *x.yyy****e\(+-zz, where the number of y's is determined by the precision (default: 6). If the precision is 0 then no decimal point is output. If the E form is used then E is printed instead of e. g or G If the exponent is less than \-4 or greater than or equal to the precision, then convert floating-point number as for %e or %E. Otherwise convert as for %f. Trailing zeroes and a trailing decimal point are omitted. % No conversion: just insert %.***

For the numerical conversions the argument being converted must be an integer or floating-point string; format converts the argument to binary and then converts it back to a string according to the conversion specifier.

DIFFERENCES FROM ANSI SPRINTF

The behavior of the format command is the same as the ANSI C **sprintf** procedure except for the following differences:

[1]

%p and **%n** specifiers are not currently supported.

[2]

For **%c** conversions the argument must be a decimal string, which will then be converted to the corresponding character value.

[3]

The **l** modifier is ignored; integer values are always converted as if there were no modifier present and real values are always converted as if the **l** modifier were present (i.e. type **double** is used for the internal representation). If the **h** modifier is specified then integer values are truncated to **short** before conversion.

SEE ALSO

sprintf(3)
string(n)

KEYWORDS

conversion specifier, format, sprintf, string, substitution

get_attached_file

Gets the real path of the attached file with the pseudonym specified.

SYNTAX

```
string get_attached_file
   [-class class_name]
   [-on object_spec]
   [-echo]
   pseudonym

class_name string


object_spec list


pseudonym string
```

ARGUMENTS

-class *class_name*

Specifies the class name of the object on which the attached file is located.

-on *object_spec*

Specifies a library or design object to which the file is to be attached. The *object_spec* value can be either a collection of exactly one object or the name pattern whose class is specified by the **-class** option. The specified object should be opened in write mode. By default, the object is simply the current design, if available; otherwise, it is determined by the opened main library.

-echo

Streams out the content of the attached file(s) to the console window if *-echo* is set.

pseudonym

Specifies a pseudonym associated with the attached file. If it is provided with a "*" when all attached files on the specified object will be listed.

DESCRIPTION

This command returns the name of the real path for the attached file. If an error occurs, the command return value is set to **0**.

EXAMPLES

The following example attaches a file to the current design and fetches the string of the real path. After the file is detached from the design, the **get_attached_file** command returns 0.

```
prompt> attach_file attach1
Attaches file default_0 to design 'top'.
1


prompt> get_attached_file default_0
/nfs/milkyway/omwx/cci/unit/mw/logic/run/design/CEL/top:2_8_tmp


prompt> detach_file default_0
Info:Detaches file 'default_0' from design 'top'.
1


prompt> get_attached_file default_0
Error: Attach file with pseudonym 'default_0' doesn't exist. (MW-026)
0
```

SEE ALSO

```
attach_file(2)
detach_file(2)
MW-026.n
```

get_attribute

Retrieves the value of an attribute on an object.

SYNTAX

```
string get_attribute
   [-class class_name]
   [-quiet]
   object_spec
   attr_name


class_name   string
object_spec  string or
object_spec  collection
attr_name    string
```

ARGUMENTS

-class *class_name*

Specifies the class name of *object_spec*, if *object_spec* is a name. The following is a list of valid values for *class_name*:

design
port
cell
pin
net
lib
lib_cell
lib_pin
clock
bound
net_shape
placement_blockage
route_guide
route_shape
terminal
text

You must use this option if *object_spec* is a name.

-quiet

Indicates that error and warning messages are not to be reported.

object_spec

Specifies a single object from which to get the attribute value. The *object_spec* must be either a collection consisting of one object, or a name that is combined with the *class_name* to find the object. If *object_spec* is a name, you must also use the **-class** option.

attr_name

Specifies the name of the attribute whose value is to be retrieved.

DESCRIPTION

Retrieves the value of an attribute on an object. The object is either a collection of exactly one object, or the name of an object. If it is a name, the **-class** option is required. The return value is a string.

EXAMPLES

In the following example, the first command defines an attribute X for cells. The second command sets the attribute to a value on all cells in this level of the hierarchy. The third command retrieves the value from one cell, combines it with the application attribute **full_name**, and creates a simple report.

```
prompt> define_user_attribute -type int -class cell X
Info:User-defined attribute 'X' on class 'cell'.
1

prompt> set_attribute [get_cells *] X 30
{"i1", "i2"}

prompt> foreach_in_collection sel [get_cells *] {
?           echo -n "On '[get_attribute $sel full_name]', "
?           echo "X =  [get_attribute $sel X]"
?         }
On 'i1', X =  30
On 'i2', X =  30
```

SEE ALSO

```
collections(2)
define_user_attribute(2)
foreach_in_collection(2)
```

```
list_attributes(2)
remove_attribute(2)
set_attribute(2)
```

get_bounds

Creates a collection of bounds from the current design.

SYNTAX

```
collection_handle get_bounds
   [-quiet]
   [-within rectangle]
   [-filter expression]
   [patterns]

expression Boolean expression


patterns list


rectangle string
```

ARGUMENTS

-quiet

Suppresses the reporting of warnings and nonterminal errors.

-within *rectangle*

Creates a collection containing all bounds within the specified rectangle.

-filter *expression*

Filters the collection with the expression value, which must be a Boolean expression based on bound attributes.

patterns

Creates a collection containing the bounds whose names match the specified patterns. The default patterns is *, which means get all bounds.

DESCRIPTION

This command creates a collection of bounds that meet the selection criteria. It returns a collection handle if one or more bounds meet the selection criteria. If no bounds match the selection criteria, it returns an empty string. You can use the **get_bounds** command at the command line prompt or as an argument to another command. You can also assign its result to a variable.

See the **collections** man page for information about working with collections.

EXAMPLES

The following example creates a new bound named *MB0* and creates a collection containing all bounds within the rectangle specified by coordinates {10 10 20 20}. It filters the collection with the expression value **type==soft**.

```
prompt> create_bound -name MB0 -bounding_box {10 10 20 20}
{"MB0"}


prompt> get_bounds -within {0 0 30 30} -filter "type==soft"
{"MB0"}
```

SEE ALSO

```
create_bound(2)
remove_bounds(2)
update_bound(2)
```

get_cells

Creates a collection of cells from the current design relative to the current instance. You can assign these cells to a variable or pass them into another command.

SYNTAX

```
collection get_cells
   [-hierarchical]
   [-filter expression]
   [--of_objects objects]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-object_id integer]
   [-all]
   [-hsc separator]
   [patterns]
```

ARGUMENTS

-hierarchical

Searches for cells level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a cell block1/adder, a hierarchical search finds it using "adder". Note that this only works on Milkyway design libraries with hierarchy preservation enabled.

-filter *expression*

Filters the collection with *expression*. For any cells that match *patterns* (or *objects*), the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also,

modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

This option makes matches case-insensitive.

-exact

Disables simple pattern matching. For use when searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** options are mutually exclusive.

-of_objects ***objects***

Creates a collection of cells connected to the specified objects. In this case, each object is either a named pin or a pin collection. The **-of_objects** and *patterns* and -object_id options are mutually exclusive; you must specify one, but not more than one. In addition, you cannot use **-hierarchical** with **-of_objects**.

-object_id integer

Get cell using its MW object id. This defaults to work in current_design.

-all

Include physical only cells, such as diodes. Default is to include only cells present in the logical hierarchy.

-hsc character

Specify the hierarchy seperator character, which will be effective for this command only. The default is "/", but may be one of "/|", "@", "^", "#", ".", "|".

patterns

Matches cell names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type cell.

The patterns, -object_id, and -of_objects arguments are mutually exclusive. If the pattern argument, -of_objects and -object_id are all absent, this is is equivalent to a pattern of "*".

DESCRIPTION

The **get_cells** command creates a collection of cells in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any cells match the *patterns* or *objects* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

If any *patterns* (or *objects*) fail to match any objects and the current design is not linked, the design automatically links.

Regular expression matching is the same as in the **regexp** Tcl command. When using **-regexp**, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin

matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_cells** command at the command prompt, or you can nest it as an argument to another command (for example, **query_objects**). In addition, you can assign the **get_cells** result to a variable.

When issued from the command prompt, **get_cells** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

The implicit query property of **get_cells** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_cells** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the cells that begin with "o" and reference an FD2 library cell. Although the output looks like a list, it is not. The output is just a display.

```
prompt> get_cells "o*" -filter "ref_name == FD2"
{"o_reg1", "o_reg2", "o_reg3", "o_reg4"}
```

The following example shows that, given a collection of pins, you can query the cells connected to those pins.

```
prompt> set pinsel [get_pins o*/CP]
{"o_reg1/CP", "o_reg2/CP"}
prompt> query_objects [get_cells -of_objects $pinsel]
{"o_reg1", "o_reg2"}
```

The following example removes the wire load model from cells i1 and i2.

```
prompt> remove_wire_load_model [get_cells {i1 i2}]
Removing wire load model from cell 'i1'.
Removing wire load model from cell 'i2'.
1
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_pins(2)
link_design(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)
```

get_layer_attribute

Queries layer attribute.

SYNTAX

```
string get_layer_attribute
   [-quiet]
   [-layer layer]
   [attribute]
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-layer *layer*

Specify the layer to query attributes. The layer could be specified by a collection or by layer name.

attribute

Specify which attribute to query. The name of the attributes can be queried by: prompt>list_attributes -application -class layer

DESCRIPTION

This command retrieves the attribute value of certain layer. The return value is a Tcl string.

EXAMPLES

The following example queries the layer number.

```
> get_layer_attribute -layer M1 layer_number
3
```

SEE ALSO

```
get_layers(2)
get_attribute(2)
```

get_layers

Creates a collection of one or more layers.

SYNTAX

```
collection get_layers
   [-filter expression]
   [-quiet]
   [-regexp [-nocase]] | [-exact]
   [-include_system]
   [patterns]
   [-exact]
```

ARGUMENTS

-filter *expression*

Filters the collection with the value of expression. For any layers that match, the expression is evaluated based on the lay er's attributes. If the expression evaluates to true, the layer is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the patterns argument as real regular expressions rather than simple wildcard patterns. This option modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The -regexp and -exact options are mutually exclusive.

-nocase

Makes matches case-insensitive. You can use this option only when you also use the -regexp option.

-include_system

Includes the system layers into the collection. Without this option, just user layers defined in technology file can be included into the collection.

Layer numbers from 1 to 187 are kept for user layers, and the layer numbers great than 187 are for

system layers.

patterns

Matches layer names against patterns. Patterns can include the wildcard characters * (asterisk) and ? (question mark) or regular expressions, based on the -regexp option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * (asterisk) and ? (question mark) wildcard characters. The -regexp and -exact options are mutually exclusive.

DESCRIPTION

This command creates a collection of layers as defined in the technology file for the current library, or a collection also with system layers included if the option -include_system specified.

You can use the get_layers command at the command prompt, or you can use it as an argument nested in another command (for example, in the query_objects command) or assign its result to a variable.

EXAMPLES

The following example returns a layer for specified name:

```
> get_layers M1R
{M1R}
```

The following example returns layers for routing:

```
> get_layers -filter is_routing_layer==TRUE
{METAL5 METAL METAL2 METAL3 METAL4 M6 M7 M8 PA}
```

SEE ALSO

```
get_layer_attribute(2)
collections(2)
filter_collection(2)
query_objects(2)
collection_result_display_limit(2)
```

get_mw_cels

Creates a collection of one or more Milkyway cels.

SYNTAX

```
collection get_mw_cels [-hierarchical]
   [-quiet]
   [-regexp [-nocase] ] | [-exact]
   [-filter expression]
   [-hierarchical]
   patterns
```

Data Types

```
   expression  string
   patterns    list
```

ARGUMENTS

-hierarchical

Searches for Milkyway cels inferred by the design hierarchy relative to the current instance. The full name of the object at a particular level must match the patterns. This option does not force an auto link.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. This option modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

Makes matches case-insensitive. You can use this option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * (asterisk) and ? (question mark) wildcard characters. The **-regexp** and **-exact** options are mutually exclusive.

-filter *expression*

Filters the collection with the value of *expression*. For any Milkyway cels that match, the expression is evaluated based on the cel's attributes. If the expression evaluates to **true**, the Milkyway cel is included in the result.

-hierarchical

Search milkyway designs level-by-level in current instance.

patterns

Matches Milkyway celnames against patterns. Patterns can include the wildcard characters * (asterisk) and ? (question mark) or regular expressions, based on the **-regexp** option. Patterns can also include collections of Milkyway celtype.

DESCRIPTION

This command creates a collection of Milkyway cels from those currently loaded into the tool that match certain criteria. It returns a collection if any Milkyway cels match the *patterns* value and pass the filter (if specified). If no objects matched the criteria, the command returns an empty string.

Regular expression matching is the same as for the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression* options. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding *.** to the beginning or end of the expressions as needed.

You can use the **get_mw_cels** command at the command prompt, or you can use the it as an argument nested in another command (for example, in the **query_objects** command) or assign its result to a variable.

When issued from the command prompt, the **get_mw_cels** command behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum by using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_mw_cels** command provides a fast, simple way to display Milkyway cels in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use the **get_mw_cels** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the mw_cels that begin with *mpu*. Although the output looks like a list, it is just a display. You can use the **list_mw_cels** command to see a complete of Milkyway cels.

```
prompt> get_mw_cels mpu*
{"mpu_0_0", "mpu_0_1", "mpu_1_0", "mpu_1_1"}
```

The following example shows that, given a collection of Milkyway cels, you can remove those cels.

```
prompt> remove_mw_cel [get_mw_cels mpu*]
Removing mw_cel mpu_0_0...
Removing mw_cel mpu_0_1...
Removing mw_cel mpu_1_0...
Removing mw_cel mpu_1_1...
```

SEE ALSO

```
collections(2)
filter_collection(2)
list_mw_cels(2)
query_objects(2)
regexp(2)
remove_mw_cel(2)
collection_result_display_limit(3)
```

get_net_shapes

Creates a collection by selecting net shapes from the current design. Returns a collection handle that can be assigned to a variable or passes them to another command.

SYNTAX

```
collection_handle get_net_shapes
   [ {-within { {ll_x ll_y} {ur_x ur_y} }
     | {-region { {ll_x ll_y} {ur_x ur_y} } }
     | {-at {x y} } ]
   [-filter expression]
   [-quiet]
   [patterns | -of_objects net_list]

ll_x         string
ll_y         string
ur_x         string
ur_y         string
expression   string
patterns     list
net_list     list
```

ARGUMENTS

-within { {*ll_x ll_y*} {*ur_x ur_y*} }

Creates a collection containing all net shapes within the specified rectangle. The **-within**, **-region**, and **-at** options are all mutually exclusive.

-region { {*ll_x ll_y*} {*ur_x ur_y*} }

Creates a collection containing all net shapes that intersect the specified rectangle.

The **-within**, **-region**, and **-at** options are all mutually exclusive.

-at {*x y*}

Creates a collection containing all net shapes that overlap the specified point.

The **-within**, **-region**, and **-at** options are all mutually exclusive.

-filter *expression*

Filters the collection with *expression*. For any net shapes that match the *patterns* argument, the expression is evaluated based on the net shape's attributes. If the expression evaluates to *true*, the net shape is included in the result.

Use the **list_attributes** command to determine the net shape attributes.

-quiet

Suppresses warning and error messages.

patterns

Matches the net shape names against the patterns in the current design. You can specify the patterns with the following format:

"*" means all net shapes; "hw*" means all horizontal wire net shape; "hw76" means one horizontal wire net shape whose object_id is 76; "vw*" means all vertical wire net shape; while "vw77" means one vertical wire net shape whose object_id is 77.

This argument and **-of_objects** are mutually exclusive.

If both this argument and *-of_objects* aren't specified, this command uses "*" as the pattern.

-of_objects *net_list*

Creates a collection containing the net shapes connected to the specified nets.

This argument and *patterns* are mutually exclusive.

DESCRIPTION

This command creates a collection of net shapes by selecting net shapes from the current design that meet the selection criteria. It returns a collection handle if one or more net shapes meet the selection criteria. If no net shapes match the selection criteria, it returns an empty string.

Use the **get_net_shapes** command as an argument to another command, or assign its result to a variable. See the example below for details.

See the **collection** command man page for information about working with collections.

EXAMPLES

The following examples create net shape collections.

```
prompt> get_net_shapes * -region {{4 20} {15 30}}
{"HW7264", "VW6766"}

prompt> get_net_shapes -of_objects n300 -region {{4 20} {15 30}}
```

```
{"VW6766"}
```

SEE ALSO

create_net_shape(2)
create_route_shape(2)
get_attribute(2)
query_objects(2)
remove_net_shape(2)

get_nets

Creates a collection of nets from the netlist. You can assign these nets to a variable or pass them into another command.

SYNTAX

```
collection get_nets
   [-hierarchical]
   [-filter expression]
   [-of_objects objects]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-object_id integer]
   [-all]
   [patterns]

expression string
patterns   list
objects    list
```

ARGUMENTS

-hierarchical

Searches for nets level by level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a net block1/muxsel, a hierarchical search would find it using muxsel.

The **-hierarchical** and **-of_objects** options are mutually exclusive.

-filter *expression*

Filters the collection with *expression*. For any nets that match *patterns* or *objects*, the expression is evaluated based on the cell's attributes. If the expression evaluates to *true*, the net is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions, rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

Makes matches case-insensitive.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** options are mutually exclusive.

-object_id integer

Get one net using its MW object id. This works in the currently open MW design.

-all

Include power and ground nets.

-of_objects *objects*

Creates a collection of nets connected to the specified objects. Each object is either a named pin or a pin collection. The **-of_objects** and *patterns* arguments are mutually exclusive. You must specify one, but not both. In addition, you cannot use **-hierarchical** with **-of_objects**.

patterns

Matches net names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type net.

The patterns, -object_id, and -of_objects arguments are mutually exclusive. If the pattern argument, -of_objects and -object_id are all absent, this is is equivalent to a pattern of "*".

DESCRIPTION

The **get_nets** command creates a collection of nets in the current design, relative to the current instance that match certain criteria. The command returns a collection if any nets match the *patterns* or *objects* and pass the filter (if specified). If no objects matched the criteria, the empty collection is returned.

If any *patterns* or *objects* fail to match any objects and the current design is not linked, the design automatically links.

Regular expression matching is the same as in the **regexp** Tcl command. When using **-regexp**, take care in the way you quote the *patterns* and filter *expression*. Use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored. The expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expression as needed.

You can use the **get_nets** command at the command prompt, or you can nest it as an argument to

another command, such as the **query_objects** command. In addition, you can assign the **get_nets** result to a variable.

When issued from the command prompt, **get_nets** behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change the maximum using the **collection_result_display_limit** variable.

The implicit query property of **get_nets** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_nets** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** command man page.

EXAMPLES

The following example queries the nets that begin with NET in block block1. Although the output looks like a list, it is just a display.

```
prompt> get_nets block1/NET
{"block1/NET1QNX", "block1/NET2QNX"}
```

The following example shows that with a collection of pins, you can query the nets connected to those pins.

```
prompt> current_instance block1
block1

prompt> set pinsel [get_pins {o_reg1/QN o_reg2/QN}]
{"o_reg1/QN", "o_reg2/QN"}

prompt> query_objects [get_nets -of_objects $pinsel]
{"NET1QNX", "NET2QNX"}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_pins(2)
link_design(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)
```

get_physical_lib_cells

Creates a collection of library cells from the libraries loaded into the tool.

SYNTAX

```
collection get_physical_lib_cells
   [-quiet]
   [-regexp | -exact]
   [-nocase]
   [-filter expression]
   [patterns | -of_objects objects]
   [-hsc separator]
```

Data Types

```
   expression     string
   patterns       list
   objects        collection
   separator      string
```

ARGUMENTS

-quiet

Suppresses information, warning, and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern.

This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

The regular expression matching is similar to the Tcl **regexp** command. When using the **-regexp** option, be careful how you quote the *patterns* argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-nocase

Makes matches case-insensitive, both for the *patterns* argument and for the ==, =~, and !~ filter operators.

-filter *expression*

Filters the collection with the specified expression. For each physical library cell in the collection, the expression is evaluated based on the physical library cell's attributes. If the expression evaluates to true, the physical library cell is included in the result.

To see the list of physical library cell attributes that you can use in the expression, use the **list_attributes -application -class physical_lib_cell** command.

For more information about how to use the **-filter** option, see the **filter_collection** man page.

patterns

Creates a collection of physical library cells whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case sensitive unless you use the **-nocase** option.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

-of_objects *objects*

Creates a collection of physical library cells that are referenced by the specified cells or that own the specified library pins. Each object can be either a cell instance or a library pin.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

-hsc *separator*

Specifies the hierarchy separator character that is used to create unambiguous names whenever a design uses a / (slash) within an object name.

You can use the following characters as hierarchy separator characters: / (slash), @ (at sign), ^ (caret), # (pound sign), . (period), and | (vertical bar).

The default is / (slash).

DESCRIPTION

This command creates a collection of library cells from libraries currently loaded into the tool that match the specified criteria.

The command returns a collection if any cells match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example queries all library cells that are in the misc_cmos library and begin with AN2. Although the output looks like a list, it is only a display.

```
prompt> get_physical_lib_cells misc_cmos/AN2*
{misc_cmos/AN2 misc_cmos/AN2P}
```

The following example shows one way to find out the name of the library cell used by a specific cell.

```
prompt> get_physical_lib_cells -of_objects [get_cells o_reg1]
{misc_cmos/FD2}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_physical_libs(2)
get_physical_lib_pins(2)
list_attributes(2)
query_objects(2)
collection_result_display_limit(3)
wildcards(3)
```

get_physical_lib_pins

Creates a collection of library cell pins from the libraries loaded into the tool.

SYNTAX

```
collection get_physical_lib_pins
   [-quiet]
   [-regexp | -exact]
   [-nocase]
   [-filter expression]
   [-all]
   [patterns | -of_objects objects]
   [-hsc separator]
```

Data Types

```
   expression     string
   patterns       list
   objects        collection
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern.

This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

The regular expression matching is similar to the Tcl **regexp** command. When using the **-regexp** option, be careful how you quote the *patterns* argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-nocase

Makes matches case-insensitive, both for the *patterns* argument and for the ==, =~, and !~ filter operators.

-filter ***expression***

Filters the collection with the specified expression. For each physical library pin in the collection, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

To see the list of physical library pin attributes that you can use in the expression, use the **list_attributes -application -class physical_lib_pin** command.

For more information about how to use the **-filter** option, see the **filter_collection** man page.

patterns

Creates a collection of physical library pins whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case sensitive unless you use the **-nocase** option.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

-of_objects ***objects***

Creates a collection of physical library pins that are referenced by the specified pins or that are owned by the specified library cells. Each object can be either a pin on a cell instance or a library cell.

Creates a collection of physical library pins that are connected to the specified objects. Each object can be either a cell instance or a library pin.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

-hsc ***separator***

Specifies the hierarchy separator character that is used to create unambiguous names whenever a design uses a slash (/) within object names.

You can use the following characters as hierarchy separator characters: slash (/), at sign (@), caret (^), pound sign (#), period (.), and vertical bar (|), with the slash (/) being the default.

-all

Includes power and ground pins.

DESCRIPTION

This command creates a collection of library cell pins from libraries currently loaded into the tool that match the specified criteria. By default, power and ground pin are not included. To include power and ground pins, you must use the **-all** option.

The command returns a collection if any pins match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example queries all pins of the AN2 library cell in the misc_cmos library. Although the output looks like a list, it is only a display.

```
prompt> get_physical_lib_pins misc_cmos/AN2/*
{misc_cmos/AN2/A misc_cmos/AN2/B misc_cmos/AN2/Z}
```

The following example shows one way to determine the library pin used by a particular pin in the netlist.

```
prompt> get_physical_lib_pins -of_objects o_reg1/Q
{misc_cmos/FD2/Q}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_physical_libs(2)
get_physical_lib_cells(2)
list_attributes(2)
query_objects(2)
collection_result_display_limit(3)
wildcards(3)
```

get_physical_libs

Creates a collection of libraries from the libraries loaded into the tool.

SYNTAX

```
collection get_physical_libs
   [-quiet]
   [-regexp | -exact]
   [-nocase]
   [-filter expression]
   [patterns | -of_objects objects]
```

Data Types

```
   expression     string
   patterns       list
   objects        collection
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern.

This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

The regular expression matching is similar to the Tcl **regexp** command. When using the **-regexp** option, be careful how you quote the *patterns* argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-nocase

Makes matches case-insensitive, both for the *patterns* argument and for the ==, =~, and !~ filter operators.

-filter *expression*

Filters the collection with the specified expression. For each physical library in the collection, the expression is evaluated based on the library's attributes. If the expression evaluates to true, the library is included in the result.

To see the list of physical library attributes that you can use in the expression, use the **list_attributes -application -class physical_lib** command.

For more information about how to use the **-filter** option, see the **filter_collection** man page.

patterns

Creates a collection of physical libraries whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case sensitive unless you use the **-nocase** option.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

-of_objects *objects*

Creates a collection of physical libraries that own the specified library cells. Each object must be a library cell.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of physical libraries from libraries currently loaded into the tool that match the specified criteria.

The command returns a collection if any libraries match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100

objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example queries all loaded libraries. Although the output looks like a list, it is only a display.

```
prompt> get_physical_libs
{misc_cmos misc_cmos_io}
```

The following example shows that you can remove the libraries in a library collection. Note that you cannot remove libraries if they are referenced by a design.

```
prompt> remove_reference_library [get_physical_libs misc*]
Removing library misc_cmos...
Removing library misc_cmos_io...
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_physical_lib_cells(2)
get_physical_lib_pins(2)
list_attributes(2)
query_objects(2)
collection_result_display_limit(3)
wildcards(3)
```

get_pin_guides

Creates a collection of pin guides that matches the selection criteria.

SYNTAX

```
collection get_pin_guides
   [-quiet]
   [-regexp | -exact]
   [-nocase]
   [-filter expression]
   [patterns | -object_id object_id]
```

Data Types

```
   expression    string
   patterns      string
   object_id     integer
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern.

This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

The regular expression matching is similar to the Tcl **regexp** command. When using the **-regexp** option, be careful how you quote the *patterns* argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-nocase

Makes matches case-insensitive, both for the *patterns* argument and for the ==, =~, and !~ filter operators.

-filter *expression*

Filters the collection with the specified expression. For each pin guide in the collection, the expression is evaluated based on the pin guide's attributes. If the expression evaluates to true, the pin guide is included in the result.

To see the list of pin guide attributes that you can use in the expression, use the **list_attributes -application -class pin_guide** command.

For more information about how to use the **-filter** option, see the **filter_collection** man page.

patterns

Creates a collection of pin guides whose full names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case sensitive unless you use the **-nocase** option.

The *patterns* and **-object_id** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

-object_id *object_id*

Creates a collection that contains the pin guide with the specified Milkyway object ID. To determine the Milkyway object ID, use the **get_attribute** command to return the value of the **object_id** attribute.

The *patterns* and **-object_id** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of pin guides from the current design that match the specified criteria.

The command returns a collection if any pin guides match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example gets all pin guides in the design.

```
prompt> get_pin_guides
```

The following example gets the pin guides that match the specified regular expression.

```
prompt> get_pin_guides -regexp {[Pp].*}
```

SEE ALSO

```
collections(2)
create_pin_guide(2)
filter_collection(2)
list_attributes(2)
query_objects(2)
report_pin_guides(2)
collection_result_display_limit(3)
wildcards(3)
```

get_pins

Creates a collection of pins from the netlist. You can assign these pins to a variable or pass them into another command.

SYNTAX

```
collection get_pins
   [-hierarchical]
   [-filter expression]
   [-of_objects objects]
   [-object_id integer]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-leaf]
   [-all]
   [-hsc separator]

   [patterns]
```

ARGUMENTS

-hierarchical

Searches for pins level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a pin block1/adder/D[0], a hierarchical search finds it using adder/D[0]. You cannot use **-hierarchical** with **-of_objects**.

-filter *expression*

Filters the collection with *expression*. For any pins that match *patterns* or *objects*, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

-of_objects *objects*

Creates a collection of pins connected to the specified objects. Each object is a named cell or net, or cell collection or pin collection. The **-of_objects** and *patterns* arguments are mutually exclusive. You must specify one, but not both. In addition, you cannot use **-hierarchical** with **-of_objects**.

-object_id integer

Get the pin with this object Id. Mutually exclusive with patterns or of_objects.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** arguments are mutually exclusive.

-nocase

When combined with **-regexp**, makes matches case-insensitive.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** arguments are mutually exclusive.

-leaf

You can use this option only with **-of_objects**. For any nets in the *objects* argument to **-of_objects**, only pins on leaf cells connected to those nets will be included in the collection. In addition, hierarchical boundaries are crossed in order to find pins on leaf cells.

-all

Include power and ground pins.

-hsc character

Specify the hierarchy separator character, which will be effective for this command only. The default is "/", but may be one of "/|", "@", "^", "#", ".", "|".

patterns

Matches pin names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type pin.

The patterns, -object_id, and -of_objects arguments are mutually exclusive. If the pattern argument, -of_objects and -object_id are all absent, this is is equivalent to a pattern of "*".

DESCRIPTION

The **get_pins** command creates a collection of pins in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any pins match *patterns* or *objects* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

When used with **-of_objects**, **get_pins** searches for pins connected to any cells or nets specified in

objects. For net objects, there are two variations of pins that will be considered. By default, only pins connected to the net at the same hierarchical level are considered. However, if an input net is a top hierarchical net inside some soft macro connected to one or more boundary pins, the returned boundary pin(s) will be the leaf pin(s) on the soft macro containing this input net. When combined with the **-leaf** option, only pins connected to the net that are on leaf cells are considered. In this case, hierarchical boundaries are crossed in order to find pins on leaf cells. Note that **-leaf** has no effect on the pins of cells.

If no *patterns* or *objects* match any objects, and the current design is not linked, the design automatically links.

When a cell has bus pins, **get_pins** can find them in several ways. For example, if cell u1 has a bus A with indexes 2 to 0, and the bus_naming_style for your design is %s[%d], then to find these pins, you can use u1/A[*] as the pattern. You can also find the same three pins with u1/A as the pattern.

Regular expression matching is the same as in the **regexp** Tcl command. When using **-regexp**, take care in the way you quote the *patterns* and filter *expression*. Use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored. The expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expression as needed.

You can use the **get_pins** command at the command prompt, or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the **get_pins** result to a variable.

When issued from the command prompt, **get_pins** behaves as though **query_objects** has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The implicit query property of **get_pins** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_pins** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** command man page.

EXAMPLES

The following example queries the CP pins of cells that begin with o. Although the output looks like a list, it is just a display.

```
prompt> get_pins o*/CP
{"o_reg1/CP", "o_reg2/CP", "o_reg3/CP", "o_reg4/CP"}
```

The following example shows that given a collection of cells, you can query the pins connected to those cells.

```
prompt> set csel [get_cells o_reg1]
{"o_reg1"}

prompt> query_objects [get_pins -of_objects $csel]
{"o_reg1/D", "o_reg1/CP", "o_reg1/CD", "o_reg1/Q", "o_reg1/QN"}
```

The following example shows the difference between getting local pins of a net and leaf pins of net. In this example, NET1 is connected to i2/a and reg1/QN. Cell i2 is hierarchical. Within i2, port a is connected to

the U1/A and U2/A.

```
prompt> get_pins -of_objects [get_nets NET1]
{"i2/a", "reg1/QN"}

prompt> get_pins -leaf -of_objects [get_nets NET1]
{"i2/U1/A", "i2/U2/A", "reg1/QN"}
```

The following example shows how to create a clock using a collection of pins.

```
prompt> create_clock -period 8 -name CLK [get_pins o_reg*/CP]
1
```

SEE ALSO

collections(2)
create_clock(2)
filter_collection(2)
get_cells(2)
link_design(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)

get_placement_blockages

Creates a collection of placement blockages from the current design. You can assign these placement blockages to a variable or pass them to another command.

SYNTAX

```
collection_handle get_placement_blockages
   [-within rectangle]
   [-touching rectangle]
   [-filter expression]
   [-quiet]
   [-type  hard | soft ]
   [patterns]

rectangle   list
expression  string
patterns    list
```

ARGUMENTS

-within rectangle

Creates a collection containing all blockages within the specified rectangle.

The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-touching rectangle

Creates a collection containing all blockages touching (touch or within) the specified rectangle.

The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-filter *expression*

Filters the collection with expression. For any placement blockages that match the patterns option, the expression is evaluated based on the attributes of the placement blockages. If the expression evaluates to *true*, the placement blockages is included in the result.

-quiet

Suppresses warning and error messages.

-type hard | soft

Indicates the type of placement blockage. If this option isn't specified, this command ignores the type of those matched placement blockages.

patterns

Specifies the placement blockages. The patterns can be a collection handle of blockages, or other formats, similar to the following:

"*" means all placement blockages; "xx*" means any placement blockage whose name begins with "xx"; "pb77" means one placement blockage who has no name yet, and its object_id is 77; while "pb*" means any placement blockage which hasn't name yet.

If this argument isn't specified, This command uses "*" as the *pattern*.

DESCRIPTION

This command creates a collection of placement blockages that meet the selection criteria. It returns a collection handle if one or more blockages meet the selection criteria. If no blockages match the selection criteria, it returns an empty string. Use the **get_placement_blockages** command as an argument to another command or assign its result to a variable. Refer to the example below for details.

See the **collection** command man page for information about working with collections.

EXAMPLES

The following examples show some uses of this command to create placement blockages.

```
prompt> get_placement_blockages * -within {{2 2} {25 25}}
{"PB5389"}

prompt> get_placement_blockages * -filter "area > 1000"
{"PB4683"}
```

SEE ALSO

```
create_placement_blockage(2)
get_attribute(2)
get_route_guides(2)
query_objects(2)
```

```
remove_placement_blockage(2)
remove_route_guide(2)
```

get_polygon_area

Calculate the area of the input polygon.

SYNTAX

```
double get_polygon_area
   polygon
```

Data Types

polygon	list

ARGUMENTS

polygon

A list of points that represents a polygon and the format is like this: {{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}. The coordinate unit is specified in technology file (usually it is micron).

DESCRIPTION

This command returns the area covered by the input polygon. The input for this command is a list of points which represents a polygon; and the returned value represents the area of the input polygons. Before this command is used, the library should be opened.

EXAMPLES

The following command returns a value represents the area of the input polygons.

```
prompt>get_polygon_area {{5 5} {20 5} {20 20} {15 20} {15 10} \
```

```
{10 10} {10 15} {5 15} {5 5}}
150.0
```

SEE ALSO

convert_to_polygon(2)
resize_polygon(2)
boolean_polygons(2)

get_ports

Creates a collection of ports from the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection get_ports
   [-filter expression]
   [-of_objects objects]
   [-object_id integer]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-all]
   [patterns]
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any ports that match *patterns*, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the cell is included in the result.

-of_objects *objects*

Creates a collection of ports connected to the specified objects. Each object is a named net collection. **-of_objects** and *patterns* are mutually exclusive. You must specify one, but not both.

-object_id integer

Get port using its MW object id. This works in the currently open MW design.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather

than simple wildcard patterns. The **-regexp** and **-exact** arguments are mutually exclusive.

-nocase

When combined with **-regexp**, makes matches case-insensitive. You can use **-nocase** only when you also use **-regexp**.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** arguments are mutually exclusive.

-all

Include power/ground ports.

patterns

Matches port names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type port.

The patterns, -of_objects, and -object_id arguments are mutually exclusive. If the pattern argument, -of_objects and -object_id are all absent, this is is equivalent to a pattern of "*".

DESCRIPTION

The **get_ports** command creates a collection of ports in the current design that match certain criteria. The command returns a collection if any ports match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using **-regexp**, take care in the way you quote the *patterns* and filter *expression*. Use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored. The expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expression as needed.

You can use the **get_ports** command at the command prompt, or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the **get_ports** result to a variable.

When issued from the command prompt, **get_ports** behaves as though **query_objects** has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The implicit query property of **get_ports** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_ports** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** command man page. In addition, refer to the man pages for the **all_inputs** and **all_outputs** commands, which also create collections of ports.

EXAMPLES

The following example queries all input ports beginning with *mode*. Although the output looks like a list, it is just a display.

```
prompt> get_ports "mode*" -filter {direction == in}
{"mode[0]", "mode[1]", "mode[2]"}
```

The following example sets the driving cell for ports beginning with *in* to an FD2.

```
prompt> set_driving_cell -cell FD2 -library my_lib [get_ports in*]
```

The following example reports ports connected to nets matching the pattern *bidir**.

```
prompt> report_port [get_ports -of_objects [get_nets "bidir*"]]
```

SEE ALSO

all_inputs(2)
all_outputs(2)
collections(2)
filter_collection(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)

get_preferred_routing_direction

Fetches the preferred routing direction on a specified layer.

SYNTAX

```
list get_preferred_routing_direction
   [-layer name]

name string
```

ARGUMENTS

-layer *name*

Specifies the routing layer name for which preferred routing direction is to be fetched.

DESCRIPTION

This command fetches preferred routing direction for the specified routing layer. If there is no layer specified, defaultly all the preferred routing direction records in current design will be returned.

If any preferred routing direction record is found, it will be inserted into the result list with the format *{layer_name direction}*.

If there is preferred routing direction record found, the command returns *0*.

EXAMPLES

The following example fetches preferred routing direction for a routing layer named *M2R*.

```
prompt> get_preferred_routing_direction -layer M2R
{{M2P vertical}}
```

The following example returns all the preferred routing direction records in current design.

```
prompt> get_preferred_routing_direction
{{M2R vertical} {M1R horizontal} {M2P vertical}}
```

SEE ALSO

```
set_preferred_routing_direction(2)
unset_preferred_routing_direction(2)
```

get_program_info

Get program information.

SYNTAX

```
string get_program_info
   -program_name
   | -install_path
   | -version
   | -dev_version
   | -sub_version
   | -integ_version
   | -full_version
   | -version_ident
```

ARGUMENTS

-program_name

Get the name of the program.

-install_path

Get the path to where Synopsys application is installed.

-version

Get the version number of the program.

-dev_version

Get the development version number of the program.

-sub_version

Get the sub version number of the program.

-integ_version

Get the integration version number of the program.

-full_version

Get the full version number of the program.

-version_ident

Get the complete information about the version of the program.

DESCRIPTION

Get various information about the current program.

EXAMPLES

The following example showes a demo result using this command against 2004.12 of Milkyway.

```
prompt> get_program_info -program_name
Milkyway
prompt> get_program_info -install_path
/remote/psd/clientstore/apf_w2004.12-sp2_prod_d
prompt> get_program_info -version
2004.12
prompt> get_program_info -dev_version
2004.12
prompt> get_program_info -sub_version
7
prompt> get_program_info -integ_version
0
prompt> get_program_info -full_version
3.5.1.7.D.INTERNAL_USE_ONLY.155
prompt> get_program_info -version_ident
Version W-2004.12-SP2-Development for IA.32 -- Mar 15, 2005
```

SEE ALSO

get_route_guides

Creates a collection of route guides from the current design. You can assign these route guides to a variable or pass them to another command.

SYNTAX

```
collection_handle get_route_guides
   [-within rectangle]
   [-touching rectangle]
   [-filter expression]
   [-quiet]
   [patterns]

   rectangle   list
   expression  string
   patterns    list
```

ARGUMENTS

-within *rectangle*

Creates a collection containing all route guides within the specified rectangle.

The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-touching *rectangle*

Creates a collection containing all route guides touching (touch or within) the specified rectangle.

The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-filter *expression*

Filters the collection with *expression*. For any route guides that match the patterns option, the expression is evaluated based on the route guides' attributes. If the expression evaluates to *true*, the route guide is included in the result.

-quiet

Suppresses warning and error messages.

patterns

Specifies the route guides. The patterns can be a collection handle of route guides, or other formats, similar to the following:

"*" or "rg*" means all route guides. While "rg7789" means one route guide whose object_id is 7789.

If this argument isn't specified, this command uses "*" as the *pattern*.

DESCRIPTION

This command creates a collection of route guides that meet the selection criteria. It returns a collection handle if one or more route guides meet the selection criteria. If no route guides match the selection criteria, it returns an empty string. Use the **get_route_guides** command as an argument to another command or assign its result to a variable. Refer to the example below for more information.

See the **collection** command man page for information about working with collections.

EXAMPLES

The following examples show some uses of this command to create route guide collections.

```
prompt> get_route_guides * -within {{2 2} {25 25}}
{"RG5389"}

prompt> get_route_guides * -filter "area > 1000"
{"RG4683"}
```

SEE ALSO

```
create_route_guide(2)
create_placement_blockage(2)
get_attribute(2)
get_placement_blockages(2)
query_objects(2)
remove_placement_blockage(2)
remove_route_guide(2)
```

get_routing_blockages

Creates a collection of routing blockages that match specified criteria.

SYNTAX

```
collection get_routing_blockages
   [-within region
     | -touching region
     | -intersect region
     | -at point]
   [-filter expression]
   [-quiet]
   [-type via | metal]
   [patterns]
```

Data Types

region list of points *point* point *expression* string *patterns* string

ARGUMENTS

-within *region*

Creates a collection that contains all routing blockages that are completely inside the specified region and do not overlap the boundary. The region boundary can be a rectangle or a rectilinear polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the polygon. The polygon must be rectilinear and the startpoint and endpoint must be the same.

The coordinate unit size is specified in the technology file, typically microns.

The **-within**, **-touching**, **-intersect**, and **-at** options are mutually exclusive; you can use no more than one. If you do not use any of these options, the command searches the entire design.

-touching *region*

Creates a collection that contains all routing blockages that touch, overlap, or are enclosed by the

specified region. The region boundary can be a rectangle or a rectilinear polygon, specified as described for the **-within** option.

-intersect *region*

Creates a collection that contains all routing blockages that intersect the boundary of the specified region, with at least part of the blockage outside of the specified region. The region boundary can be a rectangle or a rectilinear polygon, specified as described for the **-within** option.

-at *point*

Creates a collection that contains all routing blockages at the specified point. The format for specifying a point is {*x y*}.

-filter *expression*

Filters the collection with the specified expression. For each routing blockage in the collection, the expression is evaluated based on the routing blockage's attributes. If the expression evaluates to true, the routing blockage is included in the result.

Use can use the following routing blockage attributes in the expression:

Attribute name	Description
bbox	The bounding box of the routing blockage
points	Coordinates of the rectangle or polygon
name	Name of the routing blockage
layer	The name of the layer that the routing blockage is on
layer_number	The number of the layer that the routing blockage is on
area	The area of the rectangle of the routing blockage

For more information about how to use the **-filter** option, see the **filter_collection** man page.

-quiet

Suppresses warning and error messages.

-type via | metal

Specifies the type of routing blockages to query.

If you specify **-type via**, the command returns the routing blockages located on the via1Blockage-via14Blockage and polyContBlockage layers.

If you specify **-type metal**, the command returns the routing blockages located on the metal1Blockage-metal15Blockage and polyBlockage layers.

If you do not use this option, the command returns all routing blockages in the current design that match the specified criteria.

patterns

Restricts the collection to only the routing blockages whose names match the specified patterns. Each routing blockage has a name in the form of RB_*n*, where *n* is an integer. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. The default is *****, which searches for all routing blockages.

For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case-sensitive.

DESCRIPTION

This command creates a collection of routing blockages by selecting routing blockages from the current design that match specified criteria.

The command returns a collection if any routing blockages match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**; or assign the result to a variable.

When issued from the command prompt, the command behaves as though you have used the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by setting the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example returns the routing blockages within the rectangle with the lower-left corner at {2 2} and the upper-right corner at {25 25}.

```
prompt> get_routing_blockages -within {{2 2} {25 25}}
{RB_5389}
```

The following example returns the routing blockages that intersect the specified polygon.

```
prompt> get_routing_blockages  \
   -intersect {{10 5} {20 5} {20 15} {10 15} {10 5}}
{RB_4306}
```

The following example returns the routing blockages whose area is larger than 1000.

```
prompt> get_routing_blockages -filter "area > 1000"
{RB_4683}
```

The following example returns all routing blockages.

```
prompt> get_routing_blockages "RB_*"
{RB_4683 RB_5389}
```

The following example shows how to use the **get_attribute** command to query the value of routing blockage attributes.

```
prompt> get_attribute [get_routing_blockages  RB_4683] layer
via3Blockage
prompt> get_attribute [get_routing_blockages RB_4683] bbox
{0.000 0.000{ {100.000 100.000}
prompt> get_attribute [get_routing_blockages RB_4683] area
10000.000000
```

To get a complete report of the values of all attributes for a routing blockage, use the **report_attribute** command, as shown in the following example.

```
prompt> report_attribute -application [get_routing_blockages RB_4683]
Design  Object    Type   Attribute Name   Value
-------------------------------------------------------------------
CORE    RB_4683   float     area          10000.000000
CORE    RB_4683   string    bbox          {0.000 0.000} {100.000 100.000}
CORE    RB_4683   int       cell_id       3
CORE    RB_4683   string    full_name     CORE/RB_4683
CORE    RB_4683   string    layer         via3Blockage
CORE    RB_4683   int       layer_number  217
CORE    RB_4683   string    name          RB_4683
CORE    RB_4683   string    object_class  route_guide
CORE    RB_4683   int       object_id     1234
CORE    RB_4683   string    object_type   RECTANGLE
```

SEE ALSO

create_routing_blockage(2)
remove_routing_blockage(2)
collections(2)
filter_collection(2)
get_attribute(2)
report_attribute(2)
query_objects(2)
collection_result_display_limit(3)
wildcards(3)

get_selection

Returns the list of objects currently selected in the GUI or information about the selected objects.

SYNTAX

```
collection get_selection
```

DESCRIPTION

This command constructs a collection from objects that are currently selected in the GUI and returns the collection. If no objects are selected, the command returns an empty string. By default, it displays a maximum of 100 objects. You can use the **collection_result_display_limit** variable to change the maximum.

You can use this command at the command prompt, or you can nest it as an argument to another command, such as the **query_objects** command. You can also assign the **get_selection** result to a variable.

When issued from the command prompt, **get_selection** behaves as though **query_objects** has been called to display the objects in the collection. The "implicit query" property of **get_selection** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the options of the **query_objects** command (for example, if you want to display the object class), you can use **get_selection** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example assigns a collection to a variable named **collect_a**, which then is passed as an argument to the **query_objects** command.

```
prompt> set collect_a [get_selection]


{"I_PRGRM_CNT_TOP", "I_DATA_PATH", "I_REG_FILE", "I_STACK_TOP"}
```

```
prompt> query_objects $collect_a

{"I_PRGRM_CNT_TOP", "I_DATA_PATH", "I_REG_FILE", "I_STACK_TOP"}
```

SEE ALSO

change_selection(2)
collections(2)
filter(2)
filter_collection(2)
query_objects(2)
collection_result_display_limit(3)

get_terminals

Creates a collection by selecting terminals from the current design. Returns a collection handle that can be assigned to a variable or can pass them to another command.

SYNTAX

```
collection_handle get_terminals
   [-of_objects port_list]
   [-filter expression]
   [-quiet]
   [ {-regexp [-nocase] } | -exact]
   [patterns | -object_id object_id]
```

Data Types

```
port_list   list
expression  string
patterns    list
objects     list
object_id   int
```

ARGUMENTS

-of_objects *port_list*

Creates a collection of terminals that belong to the specified ports. Each element of *port_list* could be a port name, port name pattern, or port collection.

The **-of_objects**, *patterns*, and **-object_id** options are mutually exclusive.

-filter *expression*

Filters the collection with the *expression* value. For any terminals that match the *patterns* argument, the expression is evaluated based on the terminal's attributes. If the expression evaluates to **true**, the terminal is included in the result.

-quiet

Suppresses warning and error messages, if no objects match. Syntax error messages are not suppressed.

-regexp

Views the patterns argument as real regular expressions rather than simple wildcard patterns. This option modifies the behavior of the **=~** and **!~** filter operators to compare with real regular expressions, rather than simple wildcard patterns.

The **-regexp** and **-exact** options are mutually exclusive.

-nocase

Makes matches case-insensitive. You can specify the **-nocase** option only when using the **-regexp** option.

-exact

Disables simple pattern matching. Use this option when searching for objects that contain the ***** and **?** wildcard characters.

The **-exact** and **-regexp** options are mutually exclusive.

patterns

Matches the terminal names against the patterns. Patterns can include the ***** and **?** wildcard characters or regular expressions, based on the **-regexp** option. Patterns can also include collections of terminal type.

The *patterns*, **-of_objects**, and **-object_id** options are mutually exclusive.

If you do not specify both *patterns* and **-object_id**, the command uses the ***** wildcard value for the *patterns* option.

-object_id *object_id*

Specifies the object ID of one terminal.

The **-object_id**, **-of_objects**, and *patterns* options are mutually exclusive.

DESCRIPTION

This command creates a collection of terminals by selecting terminals from the current design that meet the selection criteria. It returns a collection handle if one or more terminals meet the selection criteria. If no terminals match the selection criteria, it returns an empty string.

You can use the **get_terminals** command as an argument to another command or assign its result to a variable. Refer to the example below for more information.

See the **collection** command man page for information about working with collections.

EXAMPLES

The following examples create terminal collections.

```
prompt> get_terminals [list reset clock*]
{"reset", "clock"}

prompt> get_terminals -object_id 4104
{"reset"}
```

SEE ALSO

```
create_terminal(2)
get_attribute(2)
query_objects(2)
remove_terminal(2)
```

get_texts

Creates a collection of text from the current design. You can assign these text to a variable or pass them to another command.

SYNTAX

```
collection_handle get_texts
   [-within rectangle]
   [-touching rectangle]
   [-intersect rectangle]
   [-at point]
   [-filter expression]
   [-quiet]
   [patterns]
```

Data Types

```
   rectangle   list
   point       list
   expression  string
   patterns    list
```

ARGUMENTS

-within *rectangle*

Creates a collection containing all text within the specified rectangle. The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-touching *rectangle*

Creates a collection containing all text touching (touch or within) the specified rectangle. The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-intersect *rectangle*

Creates a collection containing all text intersecting with (neither outside, nor within) the specified rectangle. The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-at *point*

Creates a collection containing all text where the specified point locates. The format of a point specification is {*x y*}.

-filter *expression*

Filters the collection with *expression*. For any text that match the patterns option, the expression is evaluated based on the text's attributes. If the expression evaluates to *true*, the text is included in the result.

-quiet

Suppresses warning and error messages.

patterns

Specifies the patterns of the text. The *patterns* can include collections of type text, and patterns with the format as shown in the following examples:

```
* or TEXT#* removes all text
TEXT#6400 removes one text whose object_id is 6400.
```

If this argument is not specified, the command uses * (asterisk) as the *pattern*.

DESCRIPTION

This command creates a collection of text that meet the selection criteria. It returns a collection handle if one or more text meet the selection criteria. If no text match the selection criteria, it returns an empty string. Use the **get_texts** command as an argument to another command or assign its result to a variable. Refer to the example below for more information.

See the **collection** command man page for information about working with collections.

EXAMPLES

The following examples show some uses of this command to create text collections.

The following example fetches the text within the specified rectangle **{{10 9} {45 15}}** and returns a new collection containing the found text:

```
prompt> get_texts * -within {{10 9} {45 15}}
{"TEXT#6401"}
The following example fetches all the text in current design,
filters the text out which text string does not equal to helloworld
and returns a new collection containing two text,
both of which has text string helloworld:

prompt> get_texts * -filter "text == helloworld"
```

```
{"TEXT#6400", "TEXT#6401"}

The following example reports warning message because there
is no text that matches the specified pattern TEXT#1234
prompt> get_text TEXT#1234
Warning: Nothing implicitly matched 'TEXT#1234' (SEL-003)
```

SEE ALSO

create_text(2)
get_attribute(2)
query_objects(2)
remove_text(2)

get_tracks

Creates a collection of track objects that match the specified criteria.

SYNTAX

```
collection get_tracks
   [-quiet]
   [-filter expression]
   [-within rectangle
      | -touching rectangle
      | -intersect rectangle
      | -at at_point]
   [patterns | -of_objects layers]
```

Data Types

```
   expression    string
   rectangle     list of points
   at_point      point
   patterns      string
   layers        collection
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the specified expression. For each track in the collection, the expression is evaluated based on the track's attributes. If the expression evaluates to true, the track is included in the result.

You can use the following track attributes in the expression:

```
   Attribute   Description

   bbox        Coordinates of the bounding box of the track object
   name        Name of the track object
   layer       Name of the layer that the track is on
```

```
direction   Direction of the wire tracks, vertical or horizontal
count       Number of parallel wire tracks in the track object
space       Pitch of the wire tracks in the track object
start       Coordinates of the lower-left corner of the object
stop        Coordinates of the upper-right corner of the object
```

For more information about how to use the **-filter** option, see the **filter_collection** man page.

-within *rectangle*

Creates a collection that contains all tracks that are completely inside the specified rectangle and do not overlap the boundary.

The format for specifying the rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, **-intersect**, and **-at** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

-touching *rectangle*

Creates a collection that contains all tracks that are inside the specified rectangle, including those that overlap the boundary.

The format for specifying the rectangle is the same as for the **-within** argument.

-intersect *rectangle*

Creates a collection that contains all tracks that intersect the boundary of the specified rectangle and at least part of the cell is outside of the specified rectangle.

The format for specifying the rectangle is the same as for the **-within** argument.

-at *point*

Creates a collection that contains all tracks at the specified point. The format for specifying a point is {*x y*}.

patterns

Creates a collection of tracks whose names match the specified patterns. Track names use the naming convention TRACK_*n*, where *n* is an integer. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case-sensitive.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify either of these arguments, the command uses * (asterisk) as the default pattern.

-of_objects *layers*

Creates a collection of tracks that are on the specified layers.

DESCRIPTION

This command creates a collection of track objects by selecting the track objects in the current design that match specified criteria.

When using the **-within**, **-touching**, or **-intersect** option, note that the behavior differs depending on the format of the specified region. An object has both a boundary and a bounding box (bbox). In some cases, these attributes differ. When you specify a region in rectangle format, the tool searches objects by their bounding box. When you specify a region in polygon format, the tool searches objects by their boundary. Searching by bounding box is faster than searching by boundary.

The command returns a collection if any tracks match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example returns the tracks within the rectangle with the lower-left corner at {2 2} and the upper-right corner at {25 25}.

```
prompt> get_tracks * -within {{2 2} {25 25}}
{TRACK_5389}
```

The following example returns the tracks that are located on the METAL2 layer.

```
prompt> get_tracks -filter "layer~=METAL2"
{TRACK_4683}
```

The following example returns all tracks.

```
prompt> get_tracks "TRACK_*"
{TRACK_4683 TRACK_5389}
```

The following example shows how to use the **get_attribute** command to query the value of a track attribute.

```
prompt> get_attribute [get_tracks TRACK_4683] layer
METAL2
prompt> get_attribute [get_tracks TRACK_4683] bbox
{{0.000 0.000} {100.000 100.000}}
```

```
prompt> get_attribute [get_tracks TRACK_4683] start
0.000 0.000
```

To get a complete report of the attribute values of all tracks, use the **report_attribute** command, as shown in the following example.

```
prompt> report_attribute -application [get_tracks TRACK_4683]
Design  Object       Type      Attribute Name   Value
-------------------------------------------------------------------------------
CORE    TRACK_4683   string    bbox             {0.000 0.000} {100.000 100.000}
CORE    TRACK_4683   int       cell_id          3
CORE    TRACK_4683   string    layer            METAL2
CORE    TRACK_4683   string    name             TRACK_4683
CORE    TRACK_4683   string    object_class     track
CORE    TRACK_4683   int       object_id        4683
CORE    TRACK_4683   string    direction        horizontal
CORE    TRACK_4683   string    start            0.000 0.0000
CORE    TRACK_4683   string    stop             100.000 100.000
CORE    TRACK_4683   int       count            51
CORE    TRACK_4683   double    space            2.000
```

SEE ALSO

```
create_track(2)
remove_track(2)
get_attribute(2)
list_attributes(2)
report_attribute(2)
collections(2)
filter_collection(2)
query_objects(2)
collection_result_display_limit(3)
wildcards(3)
```

get_user_grid

Get user grid for this session

SYNTAX

```
status
get_user_grid
   [-design]
```

ARGUMENTS

design

Specifies in which design to get the user grid information. If this option is not specified, it will get global user grid information which is used for default value of successive run of this command.

DESCRIPTION

This command get user grid for this session of the tool. The return value is a list with following form: {{x_offset y_offset} {x_step y_step}}

EXAMPLES

The following example get global user grid for this session of the tool:

```
prompt> get_user_grid
{{0 0} {0.01 0.01}}
1
```

SEE ALSO

set_user_grid(2)

get_user_shapes

Creates a collection of user shapes that match the specified criteria.

SYNTAX

```
collection get_user_shapes
   [-quiet]
   [-regexp | -exact]
   [-nocase]
   [-within region
     | -touching region
     | -intersect region]
   [-hierarchical]
   [-filter expression]
   [patterns]
```

Data Types

```
region        list of points
expression    string
patterns      list
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern.

This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

The regular expression matching is similar to the Tcl **regexp** command. When using the **-regexp** option, be careful how you quote the *patterns* argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning

or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-nocase

Makes matches case-insensitive, both for the *patterns* argument and for the ==, =~, and !~ filter operators.

-filter *expression*

Filters the collection with the specified expression. For each user shape in the collection, the expression is evaluated based on the user shape's attributes. If the expression evaluates to true, the user shape is included in the result.

To see the list of shape attributes that you can use in the expression, use the **list_attributes -application -class shape** command.

For more information about how to use the **-filter** option, see the **filter_collection** man page.

-within *region*

Creates a collection that contains all user shapes that are completely inside the specified region and do not overlap the boundary. The region boundary can be a rectangle or a polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the input polygon. A valid polygon must be rectilinear, so the startpoint and endpoint of the polygon are the same point.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, and **-intersect** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

-touching *region*

Creates a collection that contains all user shapes that are inside the specified region, including those that overlap the boundary. The region boundary can be a rectangle or a polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the input polygon. A valid polygon must be rectilinear, so the startpoint and endpoint of the polygon are the same point.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, and **-intersect** options are mutually exclusive; you can specify only one. If

you do not specify any of these options, the command searches the entire design.

-intersect *region*

Creates a collection that contains all user shapes that intersect the boundary of the specified region and at least part of the cell is outside of the specified region. The region boundary can be a rectangle or a polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the input polygon. A valid polygon must be rectilinear, so the startpoint and endpoint of the polygon are the same point.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, and **-intersect** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

-hierarchical

Searches for user shapes across design hierarchies. When you use this option, the specified patterns are considered as shape names relative to their own design hierarchy, not relative to top design. Otherwise, the specified patterns are considered as shape names relative to the top design.

This option is mutually exclusive with the options **-within**, **-touching** and **-intersect**.

patterns

Creates a collection of user shapes whose names match the specified patterns. User shapes are named based on the type of shape they are:

- Rectangle shapes are named RECTANGLE#*n*
- Trapezoid shapes are named TRAPEZOID#*n*
- Path shapes are named PATH#*n*
- Polygon shapes are named POLYGON#*n*
- Wire shapes are named WIRE#*n*

The *n* value in the shape names is an integer value.

Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case-sensitive unless you use the **-nocase** option.

If you do not specify this argument, the command uses * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of user shapes by selecting user shapes from the current design that match the specified criteria.

This command support searching across physical hierarchy, this means the blocks in the design are searched according to the input options. There are two methods to get shapes in child blocks. One method is using the shape names relative to the top level; the other method is using local name(relative to the child block) together with the option **-hierarchical**. See the **EXAMPLES** section for more detail.

The command returns a collection if any user shapes match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example queries the shapes in a specified area. Although the output looks like a list, it is only a display.

```
prompt> get_user_shapes -within {{10.5 22.3} {32.0 40}}
{RECTANGLE#100 TRAPEZOID#120 POLYGON#145}
```

The following example gets the user shapes that intersect with the specified polygon:

```
prompt> get_user_shapes \
   -intersect {{30 20} {50 20} {50 30} {40 30} {40 40} {30 40} {30 20}}
{POLYGON#9936}
```

The following example queries the shapes that have the specified route type.

```
prompt> get_user_shapes \
   -filter {route_type=="P/G Std. Cell Pin Conn"}
{PATH#1134336 PATH#1134337 PATH#1134338 PATH#1134339 PATH#1134340}
```

The following examples get the shapes inside the block referred by u1.

```
prompt> get_user_shapes u1/PATH#12345
{u1/PATH#12345}

prompt> get_user_shapes u1/HWIRE#*
{u1/HWIRE#12301 u1/HWIRE#12302 u1/HWIRE#12303}
```

The following examples get the shapes inside the block by using the option **-hierarchical**.

```
prompt> get_user_shapes PATH#12345 -hierarchical
{PATH#12345 u1/PATH#12345}

prompt> get_user_shapes HWIRE#123* -hierarchical
{HWIRE#12341 HWIRE#12342 u1/HWIRE#12301 u1/HWIRE#12302}
```

SEE ALSO

collections(2)
create_user_shape(2)
filter_collection(2)
get_net_shapes(2)
list_attributes(2)
query_objects(2)
remove_user_shape(2)
collection_result_display_limit(3)
wildcards(3)

get_via_masters

Returns a name list of contact codes (master of via) defined in the current library.

SYNTAX

```
status_value get_via_masters
   -cut_layer cut_layer
   -up_layer up_layer
   -low_layer low_layer
   patterns
```

Data Types

```
   cut_layer list
   up_layer list
   low_layer list
   patterns list
```

ARGUMENTS

-cut_layer

Create a collection containing the via_masters with specified cut layer.

-up_layer

Create a collection containing the via_masters with specified up layer.

-low_layer

Create a collection containing the via_masters with specified low layer.

patterns

Matches via master names against patterns. Patterns can include the wildcard character ***** (asterisk) and **?** (question mark).

DESCRIPTION

This command returns a name list of via master names defined in the current library. They are defined in the library's technology file.

See the man page for the **report_mw_lib** command with the **-tech** and **-output** options for details on via masters.

The name of a contact code can be used to specify the master of a via in the **create_via** command.

EXAMPLES

The following example returns the via masters whose lower layer is METAL1.

```
prompt> get_via_masters -low_layer METAL1
VIA12 VIA12FAT
1
```

The following example returns one via master, which can be used by other commands.

```
prompt> create_via -at {356 278} -no_net\
-master [get_via_masters VIA12]
{"VIA#9051"}
```

SEE ALSO

create_via(2)
report_mw_lib(2)

get_via_regions

Creates a collection of via regions from the FRAM view of the current design that match the specified criteria.

SYNTAX

```
collection get_via_regions
   [-quiet]
   [-regexp | -exact]
   [-nocase]
   [-filter expression]
   [-within region
     | -touching region
     | -intersect region
     | -at point]
   [patterns | -of_objects ports
```

Data Types

```
expression     string
region      list of points
point      point
patterns       list
ports    collection
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern.

This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

The regular expression matching is similar to the Tcl **regexp** command. When using the **-regexp** option, be careful how you quote the *patterns* argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always

anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

-nocase

Makes matches case-insensitive, both for the *patterns* argument and for the ==, =~, and !~ filter operators.

-filter *expression*

Filters the collection with the specified expression. For each via region in the collection, the expression is evaluated based on the via region's attributes. If the expression evaluates to true, the via region is included in the result.

To see the list of via region attributes that you can use in the expression, use the **list_attributes -application -class via_region** command.

For more information about how to use the **-filter** option, see the **filter_collection** man page.

-within *region*

Creates a collection that contains all via regions that are completely inside the specified region and do not overlap the boundary. The region boundary can be a rectangle or a polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the input polygon. A valid polygon must be rectilinear, so the startpoint and endpoint of the polygon are the same point.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, **-intersect**, and **-at** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

-touching *region*

Creates a collection that contains all via regions that are inside the specified region, including those that overlap the boundary. The region boundary can be a rectangle or a polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the input polygon. A valid polygon must be rectilinear, so the startpoint and endpoint of the polygon are the same point.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, **-intersect**, and **-at** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

-intersect ***region***

Creates a collection that contains all via regions that intersect the boundary of the specified region and at least part of the via region is outside of the specified region. The region boundary can be a rectangle or a polygon.

The format for specifying a rectangle is {{*llx lly*} {*urx ury*}} or {*llx lly urx ury*}, which specifies the lower-left and upper-right corners of the rectangle.

The format for specifying a polygon is {{*x1 y1*} {*x2 y2*} ... {*xN yN*} {*x1 y1*}}, where each {*x y*} pair specifies one point of the input polygon. A valid polygon must be rectilinear, so the startpoint and endpoint of the polygon are the same point.

The coordinate unit is specified in the technology file; typically the unit is microns.

The **-within**, **-touching**, **-intersect**, and **-at** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

-at ***point***

Creates a collection that contains all via regions at the specified point. The format for specifying a point is {*x y*}.

The coordinate unit is specified in the technology file; typically it is microns.

The **-within**, **-touching**, **-intersect**, and **-at** options are mutually exclusive; you can specify only one. If you do not specify any of these options, the command searches the entire design.

patterns

Creates a collection of via regions whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. For more information about using and escaping wildcards, see the **wildcards** man page. Pattern matching is case sensitive unless you use the **-nocase** option.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify either of these arguments, the command uses * (asterisk) as the default pattern.

-of_objects ***ports***

Creates a collection of via regions that belong to the specified ports.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify either of these arguments, the command uses * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of via regions by selecting via regions from the FRAM view of the current design that match the specified criteria. The command is only valid for the FRAM view of a design.

When using the **-within**, **-touching**, or **-intersect** option, the behavior depends on whether you specify the region as a rectangle or as a polygon. A via region object has both a boundary and a bounding box (bbox) and sometimes they are different. When you specify the region in rectangle format, the tool searches for objects by their bounding box. When you specify the region in polygon format, the tools search for objects by their boundary. Note that searching by bounding box is faster than searching by boundary. The command returns a collection if any via regions match the criteria. If no objects match the criteria, the command returns an empty string.

You can use this command at the command prompt or you can nest it as an argument to another command, such as **query_objects**. In addition, you can assign the result to a variable.

When issued from the command prompt, the command behaves as though you have called the **query_objects** command to report the objects in the collection. By default, it displays a maximum of 100 objects. You can change this maximum by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example creates a collection of via regions.

```
prompt> get_via_regions -of_objects [get_ports CK]
{VIA_REGION_3840 VIA_REGION_3846}
```

The following example returns the via regions within the rectangle with the lower-left corner at {5 0} and the upper-right corner at {6 1}.

```
prompt> get_via_regions -within {{5 0} {6 1}}
{VIA_REGION_3846}
```

The following example gets the via regions that intersect the specified rectangle.

```
prompt> get_via_regions -intersect {{5 0} {6 1}}
{VIA_REGION_3840}
```

SEE ALSO

collections(2)
filter_collection(2)
get_attribute(2)
get_ports(2)
list_attributes(2)
query_objects(2)
collection_result_display_limit(3)
wildcards(3)

get_vias

Creates a collection by selecting vias from the current design. Returns a collection handle that can be assigned to a variable or passes them to another command.

SYNTAX

```
collection_handle get_vias
   [-filter expression]
   [-of_objects net_list]
   [-quiet]
   [patterns]
```

Data Types

```
   expression   string
   net_list     list
   patterns    list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any vias that match the patterns option, the expression is evaluated based on the via's attributes. If the expression evaluates to *true*, the via is included in the result.

Use the **list_attributes** command to determine the via attributes.

-of_objects *net_list*

Creates a collection containing the vias connected to the specified nets.

-quiet

Suppresses warning and error messages.

patterns

Matches the via names against the patterns in the current design. You can specify the patterns with the following formats:

- * (asterisk) indicates all vias
- *via#** indicates all vias
- *via#7689* indicates one route shape whose object_id is 7689.

If both this argument and **-of_objects** are omitted, the command uses * (asterisk) as the *pattern*.

DESCRIPTION

This command creates a collection of vias by selecting vias from the current design that meet the selection criteria. It returns a collection handle if one or more vias meet the selection criteria. If no vias match the selection criteria, it returns an empty string.

Use the **get_vias** command as an argument to another command or assign its result to a variable. Refer to the example below for more information.

See the collection command man page for information about working with collections.

EXAMPLES

The following examples create via collections:

```
prompt> get_vias -of_objects n300
{"VIA6766"}
```

SEE ALSO

```
create_net_shape(2)
create_via(2)
get_attribute(2)
query_objects(2)
remove_net_shape(2)
remove_via(2)
```

get_voltage_areas

Creates a collection of voltage areas from the current design. You can assign these voltage areas to a variable or pass them to another command.

SYNTAX

```
collection_handle get_voltage_areas
   [-within w_rectangle]
   [-touching t_rectangle]
   [-intersect i_rectangle]
   [-at point]
   [-filter expression]
   [ [-of_object cell_list] | patterns]
   [-quiet]
```

Data Types

```
   w_rectangle  list
   t_rectangle  list
   i_rectangle  list
   point        list
   expression   string
   cell_list    list
   patterns     list
```

ARGUMENTS

-within *w_rectangle*

Creates a collection containing all voltage areas within the specified rectangle. The format of the *w_rectangle* specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-touching *t_rectangle*

Creates a collection containing all voltage areas touching (touch or within) the specified rectangle. The format of the *t_rectangle* specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-intersect *i_rectangle*

Creates a collection containing all voltage areas intersecting with (neither outside, nor within) the

specified rectangle. The format of the *i_rectangle* specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

-at ***point***

Creates a collection containing all voltage areas that the specified point locates. The format of the *point* specification is {*x y*}.

-filter ***expression***

Filters the collection with the *expression* value. The expression is evaluated based on the voltage areas' attributes for any voltage areas that match the patterns option. If the expression evaluates to **true**, the voltage area is included in the result.

-of_object ***cell_list***

Creates a collection containing the voltage areas that are associated with the specified cells. The **-of_objects** and *patterns* options are mutually exclusive. The default is the value of *patterns*.

patterns

Matches voltage area names against patterns in the current design. The patterns can include the wildcard characters ***** (asterisk) and **?** (question mark). This option can include collections of voltage area type.

The *patterns* and **-of_objects** arguments are mutually exclusive.

If neither the *patterns* nor **-of_objects** option is specified, the value of *patterns* defaults to the ***** setting.

-quiet

Suppresses warning and error messages.

DESCRIPTION

This command creates a collection of voltage areas that meet the selection criteria. It returns a collection handle if one or more voltage areas meet the selection criteria. If no voltage areas match the selection criteria, it returns an empty string.

You can use the **get_voltage_areas** command as an argument to another command or assign its result to a variable. Refer to the example below for more information.

See the **collection** command man page for information about working with collections.

EXAMPLES

The following example shows one way to create a voltage area collection.

```
prompt> get_voltage_areas -of_objects buffdaG1B2I1_1
{"templeoftheking"}
```

The following example shows a different way to create a voltage area collection.

```
prompt> get_voltage_areas -intersect \
{{50.000 105.000} {300.000 350}}
{"youreahero", "templeoftheking"}
```

SEE ALSO

```
create_voltage_area(2)
get_attribute(2)
query_objects(2)
remove_voltage_area(2)
```

index_collection

Creates a single element collection. For example, given a collection and an index into it, if the index is in range, the tool extracts the object at that index and creates a new collection containing only that object. The base collection remains unchanged.

SYNTAX

```
collection index_collection
   collection1
   index

collection1   collection
index         index
```

ARGUMENTS

collection1

Specifies the collection to be searched.

index

Specifies the index into the collection. Allowed values are integers from 0 to **sizeof_collection** minus 1.

DESCRIPTION

You can use the **index_collection** command to extract a single object from a collection. The result is a new collection containing that object.

The range of indexes is from 0 to one less than the size of the collection. If the specified index is outside that range, an error message is generated.

Commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same, predictable order each time. Applications that support the sorting of

collections allow you to impose a specific order on a collection.

You can use the empty string for the *collection* argument. However, by definition, any index into the empty collection is invalid. So using **index_collection** with the empty collection always generates the empty collection as a result and generates an error message.

Note that not all collections can be indexed.

EXAMPLES

The following example extracts the first object in a collection.

```
prompt> set c1 [get_cells {u1 u2}]
{"u1", "u2"}

prompt> query_objects [index_collection $c1 0]
{"u1"}
```

SEE ALSO

collections(2)
query_objects(2)
sizeof_collection(2)

list_attributes

Lists currently defined attributes.

SYNTAX

```
string list_attributes
   [-application]
   [-class class_name]

class_name   string
```

ARGUMENTS

-application

Lists application attributes and user-defined attributes.

-class *class_name*

Limits the listing to attributes of a single class. Valid classes are design, port, cell, net, etc.

DESCRIPTION

The **list_attributes** command displays an alphabetically sorted list of currently defined attributes. The attributes are divided into two categories: application-defined and user-defined. By default, **list_attributes** lists all user-defined attributes.

Using the **-application** option adds all application attributes to the listing. Since there are many application attributes, you may want to limit the listing to a specific object class using the *class_name*.

EXAMPLES

This is an example listing of some attributes defined with **define_user_attribute**.

```
prompt> list_attributes

****************************************
Report : List of Attribute Definitions
Design :
****************************************

Properties:
    A - Application-defined
    U - User-defined
    I - Importable from db (for user-defined)

Attribute Name   Object  Type      Properties  Constraints
-------------------------------------------------------------
attr_b           cell    boolean   U
attr_d           cell    double    U
attr_f           cell    float      U
attr_i           cell    int       U,I
attr_ir1         cell    int       U           0 to 100
attr_ir2         cell    int       U           >= 0
attr_ir3         cell    int       U           <= 100
attr_oo          cell    string    U           A, B, C, D
attr_s           cell    string    U
attr_s           net     string    U
```

The following example limits the listing to net attributes only.

```
prompt> list_attributes -application -class net

****************************************
Report : List of Attribute Definitions
Design :
****************************************

Properties:
    A - Application-defined
    U - User-defined
    I - Importable from db (for user-defined)

Attribute Name         Object  Type      Properties  Constraints
-----------------------------------------------------------------
area                   net     float      A
attr_s                 net     string    U
ba_capacitance_max     net     float      A
ba_capacitance_min     net     float      A
ba_resistance_max      net     float      A
ba_resistance_min      net     float      A
base_name              net     string    A
full_name              net     string    A
net_resistance_max     net     float      A
net_resistance_min     net     float      A
object_class           net     string    A
pin_capacitance_max    net     float      A
pin_capacitance_min    net     float      A
```

```
    total_capacitance_max  net     float      A
    total_capacitance_min  net     float      A
    wire_capacitance_max   net     float      A
    wire_capacitance_min   net     float      A
```

SEE ALSO

```
define_user_attribute(2)
get_attribute(2)
remove_user_attribute(2)
report_attribute(2)
set_user_attribute(2)
```

list_mw_cels

Prints the list of Milkyway cels in the current Milkyway library.

SYNTAX

```
status_value list_mw_cels
```

ARGUMENTS

None.

DESCRIPTION

The **list_mw_cels** command prints the list of Milkyway cels in the current Milkyway library. The library must be open. It also prints the following information about the Milkyway cel:

- **open** if its an open Milkyway cel
- **read_only** if it is opened in read only mode
- **current** if it is the currently active Milkyway cel.

EXAMPLES

The following example lists the Milkyway cels in the current library:

```
prompt> list_mw_cels
TCS01_PC
VIA78_array_25
TCS01_SA
```

SEE ALSO

get_mw_cels(2)

load

Not a command. Do not use this.

SYNTAX

```
load
```

ARGUMENTS

DESCRIPTION

If you want to load a scheme file, use load_scheme

EXAMPLES

NONE

SEE ALSO

load_scheme

man

Displays reference manual pages.

SYNTAX

```
string man topic

topic string
```

ARGUMENTS

topic

Specifies the subject to display. Available topics include commands, variables, and error messages.

DESCRIPTION

This command displays the online manual page for a command, variable, or error message. Users can write man pages for their own Tcl procedures and access them with the man command by setting the **sh_user_man_path** variable to an appropriate value. See the **sh_user_man_path** variable man page for details.

EXAMPLES

The following example shows how to display the manual page for the **move_objects** command.

```
prompt> man move_objects
```

The following example shows how to display the manual page for the **MWUI-011** error message.

```
prompt> man MWUI-011
```

SEE ALSO

help(2)
sh_user_man_path(3)

mem

Returns the total memory used in this run, in KB.

SYNTAX

```
integer mem
```

ARGUMENTS

None.

DESCRIPTION

This command returns the memory, in KB, of the process in which it is running.

EXAMPLES

The following example shows the use of the **mem** command.

```
prompt> set x [mem]
```

move_objects

Moves one or more cells, ports, terminals, net shapes, route guides, placement blockages, or bounds to the specified location.

SYNTAX

```
status_value move_objects
   {-delta vector | [-from from_point] -to to_point }
   objects

from_point list
objects list
to_point list
vector list
```

ARGUMENTS

-delta *vector*

Specifies the displacement for objects to move through. The **-delta** and the **-to** options are mutually exclusive. By default, the command uses no displacement.

-from *from_point*

Specifies the reference point on the object or objects to be moved. By default, the command uses the lower-left corner of the bounding box of the given object (or collection of objects) as the reference point. You can use the **-from** option only if you also specify the **-to** option.

-to *to_point*

Specifies the new location of the reference point for the object or objects. The **-delta** and the **-to** options are mutually exclusive.

objects

Specifies the objects to be moved, which can be cells, ports, terminals, net shapes, route shapes, route guides, placement blockages, or bounds.

DESCRIPTION

This command moves one or more cells, ports, terminals, net shapes, route shapes, route guides, placement blockages, or bounds to the specified location. It is often used with other Tcl commands to make the process more symbolic and convenient.

EXAMPLES

The following example moves all route guides to the location specified by the coordinates *{1400 1600}*.

```
prompt> move_objects -to {1400 1600} [get_route_guides *]
1
```

SEE ALSO

remove_objects(2)
rotate_objects(2)

open_mw_cel

Opens a Milkyway cel.

SYNTAX

```
collection open_mw_cel
   [-readonly]
   [-library library]
   [-version \flversion]
   mw_cel_name
```

Data Types

```
mw_cel_name  list
library      string
\flversion      string
```

ARGUMENTS

-readonly

Specifies to open the Milkyway cel for reading, but not for writing. You cannot modify and save the Milkyway cel.

-library *library*

Specifies the library in which the Milkyway cel exists. If you do not specify this option, the command assumes that the specified Milkyway cel exists in the current library.

If you also specified the **-readonly** option, and the *library* you specified was not opened before, the *library* will be opened as read only. The side effect is that if you want to open another Milkyway cel for writing in the same library later, you must close the *library* first, because it will still be open as read only at that time.

If a library is already open in read only mode, the Milkyway cel is opened in read only mode, even if it is not explicitly specified.

-version \flversion

Specifies the version of the design to be opened.

mw_cel_name

Specifies the name of the Milkyway cel to open.

A full Milkyway cel name is composed of three parts: the name, the view name, and the version. For example, *top.CEL;2* specifies the second version of a Milkyway cel named *top* in the *CEL* view. If you give it a simple name, such as *test*, the command assumes you are looking for the full Milkyway cel name *test.CEL;5*, if its newest version is *5*.

You can give an Milkyway cel name list or a Milkyway cel collection as the argument. This means you can open several Milkyway cels at the same time. If successful, the last opened Milkyway cel is set as the current Milkyway cel, then you can switch the current Milkyway cel among them with the **current_mw_cel** command.

DESCRIPTION

This command opens a Milkyway cel and automatically sets it as the current Milkyway cel. If you open a Milkyway cel as readonly, you can only retrieve information from it; you are not allowed to change that Milkyway cel.

EXAMPLES

The following example opens one Milkyway cel:

```
prompt> open_mw_cel DUT
{"DUT"}
```

SEE ALSO

close_mw_cel(2)
copy_mw_cel(2)
create_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
remove_mw_cel(2)
rename_mw_cel(2)
save_mw_cel(2)

open_mw_lib

Opens a Milkyway library.

SYNTAX

```
collection open_mw_lib
   [-readonly | -write_ref]
   mw_lib

mw_lib string
```

ARGUMENTS

-readonly

Specifies to open the Milkyway library only for reading. In other words, you cannot modify and save it. The **-readonly** and **-write_ref** options are mutually exclusive. The default is to open the main library read-write and all reference libraries read-only.

-write_ref

Specifies to open the reference library for writing. This option implies that the main library is opened for writing. The **-readonly** and **-write-ref** options are mutually exclusive. The default is to open the main library read-write and all reference libraries read-only.

mw_lib

Specifies to open the Milkyway library.

DESCRIPTION

This command opens a Milkyway library. The two access permissions for an opened library are read-only or read-write. Specifying the write permission implies that read permission is granted. If you specify the **-readonly** option, the command opens the main library and all reference libraries read-only. If you specify the **-write_ref** option, it opens the main library and all reference libraries read-write.

If you open the library read-only, you cannot write to it; you can only retrieve information from it. Specifically, opening the design to write in it is not permitted. If you open the library read-write, you can modify it.

The opened Milkyway library is automatically set to be the current Milkyway library.

You cannot open more than one main library at the same time. To open another main library, you must first close the previous main library.

This command returns a collection of Milkyway libraries if it succeeds.

See the man pages for the **add_reference_library** and **remove_reference_library** commands for information on adding and removing reference libraries.

EXAMPLES

The following example opens a Milkyway library named *access05* for writing in the current session.

```
prompt> open_mw_lib access05
{"access05"}
```

SEE ALSO

add_reference_library(2)
close_mw_lib(2)
copy_mw_lib(2)
create_mw_lib(2)
current_mw_lib(2)
open_mw_lib(2)
remove_reference_library(2)
rename_mw_lib(2)
report_mw_lib(2)
update_mw_lib(2)

purge_attached_file

Purges the design's attached files by creating a hard link among totally identical attached files.

SYNTAX

```
string purge_attached_file
   [-effort low | medium | high
     | [-files {pseudoname design1 design2 ... } [-force]]
   [-reset_index]
   library
```

Data Types

```
   pseudoname  string
   design1     string
   design2     string
   library     string
```

ARGUMENTS

-effort low | medium | high

Indicates how many attached files are checked and linked if they are totally identical. The **low** value indicates that the command checks only same-pseudoname attached files that belong to same design (but different versions). The **medium** value indicates that the command checks and links only same-pseudoname attached files if they are totally identical. That is, it considers different designs. The **high** value enables the command to check all the design's attached files if they are identical and can be linked together.

You can use an abbreviated format for this option by specifying **l**, **m**, or **h** for **low**, **medium**, or **high**.

The default is for no attached files to be checked for linking. The **-effort** and **-files** options are mutually exclusive.

-files {*pseudoname design1 design2* ... }

Specifies two or more attached files to check for linking. The *pseudoname* is the name that you specified in the **attach_file** command when using it with the **-pseudonym** option, or it is a default name given by that command if you did not specify the **-pseudonym** option.

You can specify the names of two or more designs. This command tries to find attached files that belong to these specified designs. The **-effort** and **-files** options are mutually exclusive.

-force

Forces two files to be linked together. The default is to not force the files to link together.

-reset_index

Indicates that this command also resets the index of all designs' attached files in the specified library. See the **EXAMPLE** section for more information. The default is to not reset the index.

library

Specifies the library in which to clean up useless temporary attached files. The default is to clean up files in all libraries.

DESCRIPTION

This command cleans up useless temporary attached files left by an abnormal exit from the Milkyway database by default. It is a good programming practice to close the library in which you want to purge attached files.

EXAMPLES

The following example cleans up useless temporary attached files in the library named *my_lib*.

```
prompt> purge_attached_file my_lib
1
```

The following example resets the attached files index. After that, *top:1_4* becomes the name for next attached file.

```
prompt> glob my_lib/CEL/top:1*
top:1  top:1_2  top:1_4  top:1_6

prompt> purge_attached_file my_lib -reset_index
1

prompt> glob my_lib/CEL/top:1*
top:1  top:1_1 top:1_2  top:1_3
```

The following example checks all designs' attached files to determine if they can be linked.

```
prompt> purge_attached_file my_lib -effort high
Linking ...
Totally 14 files are linked (1.01 Megabytes reduced).
All done
1
```

The following example shows you how two attached files are forcibly linked together by using the **-files** option. Warning: it is dangerous to link two different attached files; the information in one attached file will be lost.

```
prompt> open_design top1 -lib my_lib
{"top1"}

prompt> attach_file -pseudonym my_type file_name1
Info:Attaches file my_type to design 'top1'.
1

prompt> close_design top1
1

prompt> open_design top2
{top2"}

prompt> attach_file -pseudonym my_type file_name2
Info:Attaches file my_type to design 'top2'.
1

prompt> save_design; close_mw_lib
1

prompt> purge_attached_file my_lib -files {my_type top1 top2}
Linking ...
Totally 0 files are linked (0.00 Megabytes reduced).
All done
1

prompt> purge_attached_file my_lib -files {my_type top1 top2} \
-force
Linking ...
Totally 1 files are linked (1.01 Megabytes reduced).
All done
1
```

SEE ALSO

```
attach_file(2)
detach_file(2)
get_attached_file(2)
set_attached_file_link_mode(2)
```

query_objects

Searches for and displays objects in the database.

SYNTAX

```
string query_objects
   [-verbose]
   [-class class_name]
   [-truncate elem_count]
   object_spec

class_name   string
elem_count   int
object_spec  list
```

ARGUMENTS

-verbose

Displays the class of each object found. By default, only the name of each object is listed. With this option, each object name is preceded by its class, as in *cell:U1/U3*.

-class *class_name*

Establishes the class for a named element in the *object_spec*. Valid classes are design, cell, net, etc.

-truncate *elem_count*

Truncates display to *elem_count* elements. By default, up to 100 elements display. To see more or less elements, use this option. To see all elements, set *elem_count* to 0.

object_spec

Provides a list of objects to find and display. Each element in the list is either a collection or an object name. Object names are explicitly searched for in the database with class *class_name*.

DESCRIPTION

The **query_objects** command (a simple interface) finds and displays objects in the runtime database. The command does not have a meaningful return value; it simply displays the objects found and returns the empty string.

The *object_spec* is a list containing collections and/or object names. For elements of the *object_spec* that are collections, **query_objects** displays the contents of the collection.

For elements of the *object_spec* that are names (or wildcard patterns), **query_objects** searches for the objects in the class specified by *class_name*. The **query_objects** command does not have a predefined, implicit order of classes for which searches are initiated. If you do not specify *class_name*, only those elements that are collections are displayed. Messages are displayed for the other elements as shown in in the EXAMPLES section.

To control the number of elements displayed, use the *-truncate* option. If the display is truncated, you see the ellipsis (...) as the last element. If the default truncation occurs, a message shows the total number of elements that would have displayed (see the EXAMPLES section).

Note that the output from **query_objects** looks similar to the output from any command that creates a collection, but the result of **query_objects** is always the empty string.

EXAMPLES

The following examples show the basic usage of **query_objects**.

```
prompt> query_objects [get_cells o*]
{"or1", "or2", "or3"}

prompt> query_objects -class cell U*
{"U1", "U2"}

prompt> query_objects -verbose -class cell \
?            [list U* [get_nets n1]]
{"cell:U1", "cell:U2", "net:n1"}
```

When you omit the **-class** option, only those elements of the *object_spec* that are collections generate output. The other elements generate error messages.

```
prompt> query_objects \[list U* [get_nets n1] n*]
Error: No such collection 'U*' (SEL-001)
Error: No such collection 'n*' (SEL-001)
{"n1"}
```

When the output is truncated, you get the ellipsis at the end of the display. For the following example, assume the default truncation is 5 (it is actually 100).

```
prompt> query_objects [get_cells o*] -truncate 2
{"or1", "or2", ...}

prompt> query_objects [get_cells *]
{"or1", "or2", "or3", "U1", "U2", ...}
Output truncated (total objects 126)
```

SEE ALSO

collections(2)
get_cells(2)
get_clocks(2)
get_designs(2)
get_generated_clocks(2)
get_lib_cells(2)
get_lib_pins(2)
get_libs(2)
get_nets(2)
get_path_groups(2)
get_pins(2)
get_ports(2)
get_qtm_ports(2)
get_timing_paths(2)

read_def

Annotates the design with the data from a file in Design Exchange Format (DEF).

SYNTAX

```
int read_def
   [ -lib <library_name>]
   [ -design <design_name>]
   [ -enforce_scaling ]
   [ -allow_physical ]
   [ -no_incremental ]
   [ -advance_cell_version ]
   [ -netl_phys {incr_incr | incr_rimport
       | rimport_rimport | nochange_incr
       | nochange_rimport}]
   [ -site <site_definition_file_name> ]
   [ -core <core_site_name> ]
   [ -h_layer <list_of_h_layers> ]
   [ -turn_via_to_inst ]
   [ -inexactly_matched_via_to_inst ]
   [ -lef <lef_file_name> ]
   [ -blackbox <blackbox_ref_file_name> ]
   [ -blackbox_size <blackbox_size> ]
   [ -no_drc_special_signal_nets ]
   [ -snet_no_shape_as_user_enter ]
   [ -snet_no_shape_as_detail_route ]
   <def_file_name(s)>

string library_name
string design_name
string site_definition_file_name
string core_site_name
string lef_file_name
string def_file_name(s)
```

ARGUMENTS

-lib *library_name*

Specifies the Milkyway library in which the design cell will be annotated with the data from the input DEF file. The string may include the path of the library, otherwise the library is resolved from the current working directory. If not specified will use the current design library.

-design *design_name*

Specifies the name of the Milkyway design cell to annotate with the data from the input DEF file. If not specified will use the current top design.

-enforce_scaling

Instructs the tool to continue to annotation the information given in the DEF input regarding of possible roundoff errors due to unit conversion.

-allow_physical

Specified read_def to annotate physical only cells/nets/pins to the cell. If not specified, read_def will not create any physical only objects in the cell.

-no_incremental

Specified read_def will cleanup the exisiting physical data from the cell and annotate the physical information from the def file. If not specified will not overwrite any existing physical annotations.

-advance_cell_version

Specifies read_def will create a new version cell. If not specified, read_def will keep current cell version.

-netl_phys {incr_incr|incr_rimport|rimport_rimport|nochange_incr|nochange_rimport}

Specifies how read_def will annotate netlist information and physical information. *incr* indicates that incrementally reads the information, and issues information when a new object is added; *rimport* indicates that reset and import the information; *nochange* indicates that drop new objects and issue warnings. The default value is incr_incr.

-site *site_definition_file_name*

Specifies the Site Definition File. This file contains the core and pad site definitions, and is generated during LEF input from the site statements. Sites are created in the Milkyway design cell using these definitions. If the Site Definition File is not specified, sites are created in the Milkyway design cell using sites and tiles that are found within the library. Searching begins in the top library and proceeds through the library reference hierarchy.

-core *core_site_name*

Specifies the core site name. Only those rows, which utilize the specified core site, are created in the Milkyway design cell. If the core site is not specified, all rows are created in the Milkyway design cell.

-h_layer *list_of_h_layers*

Specifies the list of m1, m2, m3, m4 layers whose wire track direction will be set as horizontal.Tracks of higher metal layer (> m4) will be set to alternating directions with respect to m4 layer.

-turn_via_to_inst

Specifies *turn vias* are created as via cell instances. *Turn vias* are represented in DEF as a single rectangle (usually a piece of dangling metal). If not specified, *turn vias* are created as path objects.

-inexactly_matched_via_to_inst

Specifies *inexactly matched vias* are created as via cell instances. *Inexactly matched vias* are vias which match a Milkyway technology contact code's cut dimensions, but the enclosure dimensions are larger than the contact code's enclosure dimensions. If not specified, *inexactly matched vias* are

created as contact arrays and extra paths on the enclosure layers.

-lef *lef_file_name*

Specifies to import the rotated vias and design specified nondefault rule of LEF file into the design. The incremental LEF file is generared by **write_def**.

-blackbox *blackbox_ref_file_name*

Specifies *black box reference list file*. The reference list is master name based. For the instances that reference to the listed masters, if both FRAM view and CEL view reference master cannot be found, read_def will create the CEL view master cell with soft macro cell type and give it the boundary size using the input value of "blackbox_size" or default 5% of die x/y dimension.

-blackbox_size *blackbox_size*

Specifies the percentage of the die bounding box x/y dimension that the black box boundary uses. If not specified, the default value is 5. E.g. a 100x100 die will have default black box size of 5x5.

-no_drc_special_signal_nets

Specifies No DRC check for signal and clock nets in snet setion.

-snet_no_shape_as_user_enter

Specifies that the wire segments without "+ SHAPE" statement in special nets be marked with route type "user enter". If not specified, the default value of Power/Ground/Clock strap and Detailed route type will be marked in the Milkyway database based on the net type. Output def file will not see this marked attributes if the CEL is generated using read_def command. It is recommended to have a correct +SHAPE attribute in the input DEF file to avoid flow issues.

-snet_no_shape_as_detail_route

Specifies that the wire segments without "+ SHAPE" statement in special nets be marked with route type "detail route". If not specified, the default value of Power/Ground/Clock strap and Detailed route type will be marked in the Milkyway database based on the net type. Output def file will not see this marked attributes if the CEL is generated using read_def command. It is recommended to have a correct +SHAPE attribute in the input DEF file to avoid flow issues.

def_file_name(s)

Specifies the name of the file(s) from which to draw the data to annotate on to the design. Such file(s) must be in Design Exchange Format (DEF). This is a required argument.

DESCRIPTION

The read_def command reads from a DEF file the physical data associated with the current design. The tool then annotates the design with the physical information, without overwriting any existing physical annotations.

It is the users responsibility to make sure that the DEF data matches with the netlist information in the Milkyway database.

The Milkyway design library and the cell should be opened prior to calling the read_def command.

The read_def command prints a report showing number of objects read in. The number of objects in the report is the actual number of objects that have been annotated.

EXAMPLES

In the following example, the design top is annotated with the data defined in the file named *top.def*.

```
read_def -netl_phys rimport_rimport  top.def
```

SEE ALSO

write_def(2)

read_gds

Annotates the library with the data from a file in Geometry Description Standard format(GDS).

SYNTAX

```
int read_gds
   [ -lib_name <lib_name> ]
   [ -cell_version {new|overwrite_existing_cell|update_existing_cell}]
   [ -cell_type <cell_type_file_name> ]
   [ -layer_mapping <layer_mapping_file_name> ]
   [ -boundary_layer_map <layer_number>]
   [ -use_boundary_layer_as_geometry ]
   [ -ignore_undefined_layers ]
   [ -extract_instance_name_on_layer <layer_number> ]
   [ -save_mapped_layer_only ]
   [ -generate_instance_name_by_property ]
   [ -ignore_text_box ]
   [ -text_scaling_factor <scaling_factor> ]
   [ -retain_ref_libraries ]
   <gds_file_name>

string lib_name
string cell_type_file_name
string layer_mapping_file_name
string gds_file_name
```

ARGUMENTS

-lib_name *<lib_name>*

Specifies the Milkyway library in which the cells will be annotated with the data from the input GDS file. The string may include the path of the library, otherwise the library is resolved from the current working directory. If not specified, read_gds will read the data into current opened library.

-cell_version {new|overwrite_existing_cell|update_existing_cell}

Specifies how read_gds will handle the cell version when reading GDS. "*new*" indicates that read_gds will create a new version for each cell; "*overwrite_existing_cell*" indicates read_gds will overwrite the latest version of a cell already in the library with the same name as the cell being imported; "*update_existing_cell*" indicates read_gds will merge the input data with existing cell. The default value is "*new*".

-cell_type *<cell_type_file_name>*

Specifies the cell type definition file. This file lists the cells by cell types. If cell type file is not specified, read_gds will import all cells as standard cells.

-layer_mapping *<layer_mapping_file_name>*

Specifies the layer mapping file name. If layer mapping file is not specified, read_gds will keep the layer information defined in the GDS file.

-boundary_layer_map *<layer_number>*

Specifies the layer number (1-255) if you want read_gds to create the cell boundary with the geometry's boundary box on the specified layer. If not specified, the cell boundary will be generated automatically which embraces all the geometries and the TEXT objects. You can use -ignore_text_box to exclude the TEXT objects from the cell boundary calculation.

-use_boundary_layer_as_geometry

Turn this option on to keep the layer which is used as cell boundary as a geometry. If not specified, the geometry will be deleted after cell boundary is created according to it.

-ignore_undefined_layers

Turn this option on to ignore layers not defined in the technology file. If not specified, layer not defined in the technology file will also be imported into Milkyway.

-extract_instance_name_on_layer *<layer_number>*

Specifies the layer number if you want read_gds to create instance name according to the TEXT object which is on the specified layer and embraced in the cell boundary.

-save_mapped_layer_only

Turn this option on to save the objects on those GDSII layers which are specified in the layer mapping file. If not specified, read_gds will create all the objects in the source gds file.

-generate_instance_name_by_property

Turn this option on to generate the cell instance names according to the PROPERTY record in the gds file.

-ignore_text_box

Turn this option on to ignore the TEXT objects' boundary when calculating the cell boundary. If not specified, read_gds will calculate the cell boundary to embrace all objects' boundary, including TEXT.

-text_scaling_factor *<scaling_factor>*

Specifies the scaling factor for the TEXT's object size. The default value is 1.0.

-retain_ref_libraries

Turn this option on to skip creating those cells which are already in the reference libraries. If not specified, read_gds will create all the cells defined in the source gds file.

gds_file_name

Specifies the name of the file from which to draw the data to annotate on to the design. This file must be in GDS format. This is a required argument.

DESCRIPTION

Annotates the design with the data from a file in Geometry Description Standard(GDS) .

EXAMPLES

In the following example, the library "library" is annotated with the data defined in the file named "*standard.gds*".

```
read_gds -lib_name library standard.gds
```

SEE ALSO

write_gds(2)

read_icc2_frame

Create a Milkyway library from IC Compiler II FRAME data.

SYNTAX

```
int read_icc2_frame
   -technology <tech_file_name>
   -cell_lef_files <cell_lef_files>
   [-db_files <db_files>
   [-tcl_side_file <tcl_side_file_name>]
   lib_name
```

Data Types

```
   tech_file_name    string
   cell_lef_files    list
   db_files          list
   tcl_side_file_name string
   lib_name          string
```

ARGUMENTS

-technology *<tech_file_name>*

Specifies the name of the technology file for the newly created Milkyway library.

-cell_lef_files *<cell_lef_files>*

Specifies the Cell LEF input files.

-db_files *<db_files>*

Specifies the db files for updating the port information.

-tcl_side_file *<tcl_side_file_name>*

Specifies the name of a tcl side file to be sourced after the Milkyway library is created.

-library *<mw_libs>*

Specifies the name of the Milkyway library to be created.

DESCRIPTION

This command creates a Milkyway reference library using LEF files exported from IC Compiler II.

EXAMPLES

The following example creates a Milkyway library *mw_lib*. The *cell.lef* must be a LEF file that exported using *export_dc_fram* from IC Compiler II Library Manager.

```
> read_icc2_frame -technology test.tf -cell_lef_files cell.lef mw_lib
```

SEE ALSO

read_lef

Create a Milkyway reference library ready for place and route from input LEF files.

SYNTAX

```
int read_lef
   [ -lib_name <lib_name> ]
   [ -tech_lef_files <tech_lef_files> ]
   [ -cell_lef_files <cell_lef_files> ]
   [ -layer_mapping <layer_mapping_file_name> ]
   [ -overwrite_existing_tech ]
   [ -ignore_cell_geom ]
   [ -advanced_lib_prep_mode ]
   [ -cell_version <merge | overwrite | new | ignore> ]
   [ -cell_boundary by_cell_size | by_overlap_layer ]
   [ -overlap_in_block_ring_only ]

string lib_name
string tech_lef_files
string cell_lef_files
string layer_mapping_file_name
```

ARGUMENTS

-lib_name *<lib_name>*

Specifies the Milkyway library in which the library information and reference cells will be created. The string can include the path of the library.

If no path is specified, the library is resolved from the current working directory. The read_lef command creates a new reference library based on LEF information if you do not have an existing library.

-tech_lef_files *<tech_lef_files>*

Specifies the Tech input LEF files. You can also specify multiple LEF files if, for example, you have separate LEF files for technology information and antenna information.

-cell_lef_files *<cell_lef_files>*

Specifies the Cell LEF input files. You can also specify multiple LEF files if, for example, you have separate LEF files for standard cells and macros.

-layer_mapping *<layer_mapping_file_name>*

Specifies a layer mapping file that contains the LEF layer name and Milkyway layer number. If you do not specify this file, Milkyway follows the LEF layer order with the first layer in LEF mapping to layer 1 in Milkyway and the second layer in LEF mapping to layer 2 in Milkyway, and so on. Specifying a layer mapping file is recommended.

-overwrite_existing_tech

This option is not selected by default. When you select this option, all the existing technology information from the library is removed and the new technology information from the Tech LEF Files is used instead.

-ignore_cell_geom

This option is not selected by default. When you select this option, the pin, rectangle, polygon and via objects are not created. After LEF input, you must input the GDSII data to create the physical information. This option is typically used for the Tech LEF + Physical Cell GDS flow.

-advanced_lib_prep_mode

This option is not selected by default. In the default mode, read_lef automatically completes all library preparation steps and creates a Milkyway library ready for place and route.

If this option is selected, you must manually run the steps that follow Import LEF in order to create a Milkyway Library (steps such as Extract BPV, Set PR Boundary, Set Property for Multiple Height Cells, and Define Wire Track). Use this option for special cases or for advanced library preparation.

-cell_version *<merge* | *overwrite* | *new* | *ignore>*

Merge with Existing Cell: This option is selected by default. When selected, the latest cell version will include the information defined in the LEF file and the existing database information. Cell-related information such as macro pin and macro obstructions can be appended to the existing information.

Overwrite Existing Cell: This option is not selected by default. If this option is selected, the latest cell versions will be overwritten by the LEF macro cells.

Make New Cell Version: This option is not selected by default. If this option is selected, the new cell versions will be created by the macro cells defined in LEF. The old version will be maintained but a new version based on the LEF information will be created.

Ignore LEF Cell: This option is not selected by default. If this option is selected, macros defined in the LEF file with the same name will be ignored when you import the LEF file.

-cell_boundary by_cell_size | by_overlap_layer

Specifies whether to extract rectangular or rectilinear cell boundaries for macro cells. With the **by_cell_size** option setting (the default behavior), a rectangular cell boundary is derived from the SIZE parameter in the LEF file. With the **by_overlap_layer** option setting, a rectilinear cell boundary (possibly non-rectangular) is derived from the OVERLAP layer in the LEF file.

-overlap_in_block_ring_only

Specifies whether the **-cell_boundary by_overlap_layer** setting, which selects rectilinear boundary extraction, applies to all macro cells or only BLOCK and RING type macro cells. The default is to perform rectilinear boundary extraction on all macros. With the **-overlap_in_block_ring_only** option, rectilinear boundary extraction is performed only on BLOCK and RING type macro cells. The **-cell_boundary by_overlap_layer** option has an effect only when **-cell_boundary**

by_overlap_layer is used.

DESCRIPTION

The **read_lef** command create a Milkyway reference library ready for place and route from input LEF files.

EXAMPLES

The following example read LEF files to create a milkyway library "*design*".

```
read_lef -lib_name design -tech_lef_files tech.lef -cell_lef_files cell.lef
```

SEE ALSO

auNLOApi(2)
write_lef(2)

read_oasis

Annotates the library with the data from a file in Open Artwork System Interchange Standard (OASIS).

SYNTAX

```
int read_oasis
   [ -lib_name <lib_name> ]
   [ -cell_version {new | overwrite | overwrite_confirm | merge}]
   [ -cell_type <cell_type_file_name> ]
   [ -layer_mapping <layer_mapping_file_name> ]
   [ -use_geom_on_layer_as_boundary <layer_number>]
   [ -ignore_undefined_layers ]
   [ -extract_instance_name_on_layer <layer_number> ]
   [ -generate_instance_name_by_property ]
   <oasis_file_name>

string lib_name
string cell_type_file_name
string layer_mapping_file_name
string oasis_file_name
```

ARGUMENTS

-lib_name *<lib_name>*

Specifies the Milkyway library in which the cells will be annotated with the data from the input OASIS file. The string may include the path of the library, otherwise the library is resolved from the current working directory. If not specified, read_oasis will read the data into current opened library.

-cell_version {new|overwrite|overwrite_confirm|merge}

Specifies how read_oasis will handle the cell version when reading OASIS. "*new*" indicates that read_oasis will create a new version for each cell; "*overwrite*" indicates read_oasis will overwrite the latest version of a cell already in the library with the same name as the cell being imported; "*overwrite_confirm*" indicates that read_oasis will require confirmation before it overwrites a cell. "*Merge*" indicates read_oasis will merge the input data with existing cell. The default value is "*new*".

-cell_type *<cell_type_file_name>*

Specifies the cell type definition file. This file lists the cells by cell types. If cell type file is not specified, read_oasis will import all cells as standard cells.

-layer_mapping *<layer_mapping_file_name>*

Specifies the layer mapping file name. If layer mapping file is not specified, read_oasis will keep the layer information defined in the OASIS file.

-use_geom_on_layer_as_boundary *<layer_number>*

Specifies the layer number (1-255) if you want read_oasis to create the cell boundary with the geometry's boundary box on the specified layer. If not specified, read_oasis will create a cell boundary to embrace all the objects in the cell.

-ignore_undefined_layers

Turn this option on to ignore layers not defined in the technology file. If not specified, layer not defined in the technology file will also be imported into Milkyway.

-extract_instance_name_on_layer *<layer_number>*

Specifies the layer number if you want read_oasis to create instance name according to the TEXT object which is on the specified layer and embraced in the cell boundary.

-generate_instance_name_by_property

Turn this option on to generate the cell instance names according to the PROPERTY record in the OASIS file.

oasis_file_name

Specifies the name of the file from which to draw the data to annotate on to the design. This file must be in OASIS format. This is a required argument.

DESCRIPTION

Annotates the design with the data from a file in Open Artwork System Interchange Standard(OASIS) .

EXAMPLES

In the following example, the library "Reference" is annotated with the data defined in the file named "*standard.oasis*".

```
read_oasis -lib_name Reference  standard.oasis
```

SEE ALSO

write_oasis(2)

read_verilog

Reads in one or more design or library files in Verilog format.

SYNTAX

```
status read_verilog
   [-dirty_netlist]
   [-allow_black_box]
   [-verbose]
   [-bus_direction_for_undefined_cell connection | msb | lsb]
   [-keep_module keep_module_list]
   [-top top_module_name]
   [-cell cell_name]
   verilog_files
```

Data Types

keep_module_list	list
top_module_name	string
cell_name	string
verilog_files	list

ARGUMENTS

-dirty_netlist

Controls whether the Verilog reader handles a dirty netlist. A dirty netlist is a Verilog netlist that contains objects that are inconsistent with the design libraries, such as floating pins, nets, and pins or ports. When the Verilog reader reads in a dirty netlist, it checks for the following missing or incomplete information, issues warning messages, and then creates objects that allow the tool to continue:

```
* Missing reference cell in the reference library
  There is no reference cell available in the reference
  libraries and the definition is not in the input Verilog
  file.

* Port definition is inconsistent with the reference library
  There are additional ports present that are not available
  in the reference library.

* Port definition is inconsistent with the reference cell
  There are additional ports present that are not available
  in the reference cell definition.
```

-allow_black_box

Allows the tool to continue if a module cannot be found in the reference libraries or in the input Verilog file.

If you do not specify this option, **read_verilog** issues an error message and quits when it finds an undefined module.

-verbose

Displays verbose information including possible top module names, black box names, and module names with mismatched ports.

-bus_direction_for_undefined_cell connection | msb | lsb

Specifies the bus direction for an undefined cell. If an undefined module is instantiated, **read_verilog** determines the bus direction according to one of following three factors: connection, msb or lsb. By default, it determines the bus direction from the connection.

-keep_module ***keep_module_list***

Specifies the list of keep modules. For each module in the list, the Verilog reader does not expand it into the top cell. Instead, it expands it into a soft macro cell and instantiates it in the top cell. This is for the design planning top-down flow.

-top ***top_module_name***

Specifies the top module name. The top module is the module that is not instantiated by any other modules. Generally there is only one top module in a design and **read_verilog** can scan and identify the top module automatically. However, if the design has multiple top modules, you must use this option to specify one of them.

-cell ***cell_name***

Specifies the cell name. The tool saves the design in the Milkyway database using the specified cell name. If not specified, the tool uses the name of the top module as the cell name.

verilog_files

Specifies the name of one or more Verilog files to be read.

DESCRIPTION

.prod icc

The **read_verilog** command loads in netlist information from one or more Verilog files.

SEE ALSO

rebuild_mw_lib

Rebuilds the Milkyway library.

SYNTAX

```
status_value rebuild_mw_lib
   libName
```

Data Types

```
   libName  string
```

ARGUMENTS

libName

Specifies the Milkyway library to be rebuilt. The value of *mw_lib* should be a valid library name. The library must be closed for this command to work.

DESCRIPTION

This command rebuilds a Milkyway library by scanning all designs in the associated library directory.

The command returns a status indicating success or failure.

EXAMPLES

The following example rebuilds the current Milkyway library:

```
prompt> rebuild_mw_lib
```

```
Scanning library...
    place.CEL;5
    place.CEL;4
    place.CEL;3
    place.CEL;2
    place.CEL;1
    place.EXP;1
    place.NETL;1
    place.PARA;1
Rebuilding library...
A total of 8 items have been rebuilt. (0 removed, 0 new added)
1
```

SEE ALSO

close_mw_lib(2)
copy_mw_lib(2)
create_mw_lib(2)
current_mw_lib(2)
open_mw_lib(2)
rename_mw_lib(2)
report_mw_lib(2)

remove_antenna_rules

Deletes all of the antenna rules stored in the library.

SYNTAX

```
status_value remove_antenna_rules
   [mw_lib | -lib lib_id]

mw_lib list


lib_id string
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be updated. The value of *mw_lib* can be a library name or a one-element collection of a library. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-lib *lib_id*

Specifies the ID of the Milkyway library to be updated. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

DESCRIPTION

This command deletes all of the antenna rules stored in the library. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> remove_antenna_rules
```

SEE ALSO

define_antenna_rule(2)
define_antenna_layer_ratio_scale(2)
define_antenna_layer_rule(2)
report_antenna_rules(2)

remove_attachment_file

Removes the attachment files from the specified design or from all designs.

SYNTAX

```
status remove_attachment_file
   -design design_name | -all
   [-check_only]
```

Data Types

```
   design_name          string
```

ARGUMENTS

-design *design_name*

Removes the attachment files from the specified design.

By default, when you set this option, this command removes the attachment files from all versions of the specified design. To remove the attachment files from a specific version of the design, use the following format to specify the version: "*design_name*;*version_number*".

This option is mutually exclusive with the **-all** option; you must specify one of these options.

-all

Removes the attachment files from all versions of all designs.

This option is mutually exclusive with the **-design** option; you must specify one of these options.

-check_only

Reports which attachment files will be removed, but does not actually remove them.

DESCRIPTION

This command removes the attachment files from the specified design or from all designs. It reports the following information for each removed attachment file:

- The attachment file name
- The attachment file size
- The total number of removed attachment files for each design
- The total freed disk space

Before you run this command, you must open the associated Milkyway design library. The design or designs from which you want to remove the attachment files must not be open; otherwise, the tool issues the MWUI-070 error message.

Currently this command removes only congestion map attachment files that are generated by routing commands. After removing these files, you must regenerate the congestion map before running any commands that use the congestion map.

EXAMPLES

The following example reports the attachment files that would be removed from all versions of the ROUTED design in the current Milkyway design library, but does not actually remove these files.

```
prompt> open_mw_lib design
prompt> remove_attachment_file -check_only -design ROUTED

Checking on attachment files that will be deleted in design ROUTED.CEL;1
File Name               File Size
ROUTED:1_44             519628
ROUTED:1_43             519628
ROUTED:1_42             519628
ROUTED:1_41             519628
ROUTED:1_40             519628
ROUTED:1_39             519628
ROUTED:1_38             519628
Total 7 attachment files will be removed from design ROUTED.CEL;1

Checking on attachment files that will be deleted in design ROUTED.CEL;2
File Name               File Size
ROUTED:2_44             519628
ROUTED:2_43             519628
ROUTED:2_42             519628
ROUTED:2_41             519628
ROUTED:2_40             519628
ROUTED:2_39             519628
ROUTED:2_38             519628
```

```
Total 7 attachment files will be removed from design ROUTED.CEL;2

Total 7274792 bytes disk space will be freed.
1
```

The following example reports the attachment files that would be removed from version 1 of the ROUTED design in the current Milkyway design library, but does not actually remove these files.

```
prompt> open_mw_lib design
prompt> remove_attachment_file -check_only -design "ROUTED;1"

Checking on attachment files that will be deleted in design ROUTED.CEL;1.
File Name               File Size
ROUTED:1_44             519628
ROUTED:1_43             519628
ROUTED:1_42             519628
ROUTED:1_41             519628
ROUTED:1_40             519628
ROUTED:1_39             519628
ROUTED:1_38             519628
Total 7 attachment files will be removed from design ROUTED.CEL;1.

Total 3637396 bytes disk space will be freed.
1
```

The following example removes all attachment files from all designs in the current Milkyway design library.

```
prompt> open_mw_lib design
prompt> remove_attachment_file -all

Deleting following attachment files in design ROUTED.CEL;1
File Name               File Size
ROUTED:1_44             519628
ROUTED:1_43             519628
ROUTED:1_42             519628
ROUTED:1_41             519628
ROUTED:1_40             519628
ROUTED:1_39             519628
ROUTED:1_38             519628
Total 7 attachment files were removed from design ROUTED.CEL;1

Deleting following attachment files in design ROUTED.CEL;2
File Name               File Size
ROUTED:2_44             519628
ROUTED:2_43             519628
ROUTED:2_42             519628
ROUTED:2_41             519628
ROUTED:2_40             519628
ROUTED:2_39             519628
ROUTED:2_38             519628
Total 7 attachment files were removed from design ROUTED.CEL;2

Deleting following attachment files in design icc_test_09.CEL;1
File Name               File Size
icc_test_09:1_44        519628
icc_test_09:1_43        519628
icc_test_09:1_42        519628
icc_test_09:1_41        519628
icc_test_09:1_40        519628
icc_test_09:1_39        519628
icc_test_09:1_38        519628
Total 7 attachment files were removed from design icc_test_09.CEL;1

Deleting following attachment files in design fill_test.CEL;1
```

```
File Name               File Size
Total 0 attachment files were removed from design fill_test.CEL;1

Total 10912188 bytes disk space was freed.
1
```

SEE ALSO

report_milkyway_version(2)
report_critical_area(2)
read_antenna_violation(2)
route_global(2)
route_group(2)
route_auto(2)
route_opt(2)
report_congestion(2)
route_fp_proto(2)

remove_attribute

Removes an attribute from the specified list of objects.

SYNTAX

```
collection remove_attribute
   [-class class_name]
   object_list
   attribute_name
   [-quiet]

class_name string
object_list list
attribute_name string
```

ARGUMENTS

-class *class_name*

Specifies the class name for the object specified in *object_list*, if the element of *object_list* is a name. The following is a list of valid values for *class_name*:

design
port
cell
pin
net
lib
lib_cell
lib_pin
clock
bound
net_shape
placement_blockage
route_guide
route_shape
terminal
text

object_list

Specifies a list of objects from which the attribute is to be removed. Each element in the list is either a collection or a pattern that is combined with the *class_name* to find the objects.

attribute_name

The name of the attribute to be removed.

attribute_name

Specifies the name of the attribute to be removed.

-quiet

Turns off the warning message that would otherwise be issued if the attribute or objects are not found.

DESCRIPTION

This command removes the attribute from the specified objects. For a complete listing of attributes, refer to the **attributes** man page.

This command creates a collection of objects that have the specified attribute removed. A returned empty string indicates that no object has been removed.

EXAMPLES

The following example shows the first command defining a new attribute named *X* on the net. The second command sets *30* to *X* on all the nets in the current hierarchy. The third command removes the attribute *X* from nets named *VSS* and *VDD*. The fourth command retrieves the attribute *X* from net named *VDD*. The fifth command retrieves the attribute *X* from a net named *in*.

```
prompt> define_user_attribute -type int -class net X
Info:User-defined attribute 'X' on class 'net'.
1
prompt> set_attribute [get_nets *] X 30
{"out", "in", "VDD", "VSS"}
prompt> remove_attribute [get_nets V*] X
{"VDD", "VSS"}
prompt> get_attribute [get_nets VDD] X
ERROR : Failed to get attr(X)'s value.
prompt> get_attribute [get_nets in] X
30
```

SEE ALSO

```
collections(2)
define_user_attribute(2)
get_attribute(2)
```

```
list_attributes(2)
set_attribute(2)
```

remove_base_arrays

Removes base arrays from the *current design*.

SYNTAX

```
status_value remove_base_arrays [-all | pattern]

pattern  string
```

ARGUMENTS

-all

Specifies to remove all base arrays from the current design.

pattern

Specifies the pattern of base arrays to be removed.

DESCRIPTION

This command removes base arrays from the *current design*. If the **-all** option is specified, all base array records are removed from the design. If a pattern of base arrays is given, the base arrays matched with the specified pattern are removed. The command returns a value of **1** if it succeeds, or **0** if the command fails.

EXAMPLES

The following example removes all base arrays in the design:

```
prompt> remove_base_arrays -all
1
```

SEE ALSO

create_base_array(2)
report_base_arrays(2)

remove_cell

Removes a list of cells.

SYNTAX

```
status_value remove_cell
   [-verbose]
   cell_list | -master master_list | -all

cell_list   list
master_list list
```

ARGUMENTS

-verbose

Prints more messages.

cell_list

Specifies a list of cells to remove.

-master *master_list*

Specifies a list of cell masters of which cells are to be removed. All cells of the specified master will be removed, and is the master itself.

-all

Removes all cells of the *current_design*.

DESCRIPTION

Removes a list of cells from the specified or current design. Specified masters will be removed first, and then all cells of them.

EXAMPLES

The following example removes specified cells.

```
prompt> get_cells or2t*
{"or2t_1", "or2t_2"}

prompt> remove_cell or2t*
1
```

The following example removes cells by their master.

```
prompt> get_cells *
{"U1"}

prompt> get_att [get_cells *] ref_name
INV2T

prompt> remove_cell -master INV2T
1

prompt> get_cells *
Warning: No cells matched '*' (SEL-004)
Error: Nothing matched for cells (SEL-005)
```

SEE ALSO

create_cell(2)
get_cells(2)
replace_cell(2)

remove_collections

Removes all of the collections currently present in the tool.

SYNTAX

```
<status> remove_collections
```

ARGUMENTS

None.

DESCRIPTION

The **remove_collections** command is used to remove all of the collections at once.

Since there may be some memory overhead associated with collections, it is useful to remove them before launching a large, memory consuming operation.

EXAMPLES

The following is an example of using **remove_collections**.

```
prompt> remove_collections
```

SEE ALSO

collections(2)
filter_collection(2)
query_objects(2)
report_collections(2)

remove_from_collection

Removes objects from a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection remove_from_collection
   base_collection

base_collection    x1collection
object_spec        list
```

ARGUMENTS

base_collection

Specifies the base collection to be copied to the result collection. Objects matching *object_spec* are removed from the result collection.

object_spec

Specifies a list of named objects or collections to remove. The object class of each element in this list must be the same as in the base collection. If the name matches an existing collection, the collection is used. Otherwise, the objects are searched for in the database using the object class of the base collection.

DESCRIPTION

The **remove_from_collection** command removes elements from a collection, creating a new collection.

If the base collection is homogeneous, any element of the *object_spec* that is not a collection is searched for in the database using the object class of the base collection. If the base collection is heterogeneous, then any element of the *object_spec* that is not a collection is ignored.

If nothing matches the *object_spec*, the resulting collection is a copy of the base collection. If everything in *base_collection* matches the *object_spec*, the result is the empty collection.

For background on collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example from PrimeTime gets all input ports except CLOCK.

```
prompt> set cPorts [remove_from_collection [all_inputs] CLOCK]
{"in1", "in2"}
```

SEE ALSO

add_to_collection(2)
collections(2)
query_objects(2)

remove_lib_cel

Removes all CEL views except tiles.

SYNTAX

```
int remove_lib_cel
   [ -lib_name <lib_name> ]
```

ARGUMENTS

-lib_name lib_name

Specifies the Milkyway library whose CEL views are removed.

DESCRIPTION

The **remove_lib_cel** command removes all the CEL views except tiles in the library to reduce the size of the library. This is useful for users creating front end library kits, where the CEL views are not needed for front end tools. It is recommended that this command be used after library preparation is complete and after check_library is used to check the quality of the library.
A status indicating success or failure is returned.

EXAMPLE

The following example removes all the CEL views except tiles from the specified library test.mw.

```
prompt> remove_lib_cel -lib_name test.mw
```

```
CEL views in library test.mw have been removed.
1
```

SEE ALSO

remove_mw_cel(2)
remove_design(2)

remove_mw_bounds

Removes bounds from the *current design*.

SYNTAX

```
status_value remove_mw_bounds
   [-all]
   [-name bound_name_list]
   bounds
```

Data Types

```
   bound_name_list  string
   bounds           string
```

ARGUMENTS

-all

Specifies to remove all bounds from the current design.

-name *bound_name_list*

Specifies to remove bounds with the given names.

bounds

Specifies the bounds to be removed.

DESCRIPTION

This command removes bounds from the *current design*. If the **-all** option is specified, all bounds are removed from the design. If a list of bound names is given, the bounds with the specified names are removed. The command returns a value of **1** if it succeeds, or **0** if the command fails.

EXAMPLES

The following example removes all bounds from the *current design*:

```
prompt> remove_mw_bounds [get_bounds *]
1
```

SEE ALSO

```
create_mw_bound(2)
get_bounds(2)
update_mw_bound(2)
```

remove_mw_cel

Removes Milkyway designs from the design library.

SYNTAX

```
status remove_mw_cel
   [-hierarchy [-check_only] ]
   [-version_kept count]
   [-verbose]
   [-all_versions]
   [-all_view]
   mw_cel_list
```

Data Types

```
   count           integer
   mw_cel_list     list
```

ARGUMENTS

-hierarchy

Deletes both the specified top-level design and all of its subdesigns. By default, the command deletes only the specified designs and retains subdesigns

This option can not work with **-all_views** and **-all_versions**. If **mw_cel_list** option is used with this option, it can only contain one Milkyway design. Moreover, the version number and view name of the Milkyway design will be ignored. As for the view name, the hierarchical remove can only operate on CEL view. As for the version number, hierarchical remove can only remove all versions or retain last version, which is controlled by **-version_kept** option.

-check_only

Prints the list of Milkyway designs, versions, and views, but does not delete any data. If you are unsure about the result of the command, run this option first, before performing the purge operation. This option is valid only when used with the **-hierarchy** option.

-version_kept *count*

Specifies how many recent versions to keep. The value must be equal or greater than **0**. Value **0** removes all versions. When used with the **-hierarchy** option, the only valid values are **0** or **1**.

The default behavior differs for hierarchical removal (specified by using the **-hierarchy** option) and nonhierarchical removal. For hierarchical removal, all versions are deleted if you do not specify the **-version_kept** option. For nonhierarchical removal, only the latest version is deleted if you do not specify the **-version_kept** option.

-verbose

Prints the possible cause when the command fails to remove a specified Milkyway design. Currently, this option is effective only when you use it with the **-hierarchy** option.

-all_versions

Removes all versions of specified Milkyway designs. This option will be ignored when **-hierarchy** option used.

-all_view

Removes all views of specified Milkyway designs. By default, the command removes only the specified view. This option will be ignored when -hierarchy option used.

mw_cel_list

Specifies the Milkyway designs to remove. If you also use the **-hierarchy** option with this option, the designs to remove must be top-level designs. You can specify a Milkyway design by name, pattern, or collection. For more information, see the EXAMPLES section, below.

When use with **-hierarchy** option, only one Milkyway design can be specified. And the version number or view name of the Milkyway design will be ignored. The view name is always CEL view and the version number is controlled by **-version_kept** option.

When not use with **-hierarchy** option, only the specified view and version will be deleted.

DESCRIPTION

This command removes specified Milkyway designs from the current library. Before removing designs, you must make sure that the designs are closed.

The **-version_kept** option allows you to remove older versions of the Milkyway design while keeping newer ones (to free up disk space).

The **-hierarchy** option allows you to remove or purge a Milkyway design and its whole hierarchical tree.

NOTE: Please be aware that the behavior of **remove_mw_cel** differs depending on whether you specify the **-hierarchy** option. The default command behavior deletes the specified version of the Milkyway design, but when called with the **-hierarchy** option, the command deletes all versions or non-latest versions of the Milkyway designs controlled by **-version_kept**.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example removes the latest version of the Milkyway designs whose names start with *test*.

```
prompt> get_mw_cels test*
{"test1", "test2"}
prompt> remove_mw_cel test* -verbose
Removed mw_cel test1.
Removed mw_cel test2.
1
```

The following example removes version **1** of a design named *test1*. In this case, *test1* has two versions. After removing one version, you can still get a Milkyway design collection based on its remaining version.

```
prompt> remove_mw_cel "test1.CEL;1" -verbose
Removed mw_cel test1.CEL;1.
1
prompt> get_mw_cels *
{"test1"}
```

The following example removes the design named *chip_top* and all of its subdesigns from the current design library.

```
prompt> remove_mw_cel -hierarchy chip_top
Deleting the cell: nand_macro ...
Deleting the cell: chip_top ...
1
```

The following example purges the old versions of the design named *chip_top* and all of its subdesigns from the current design library. Only the latest version of these designs is kept.

```
prompt> remove_mw_cel -hierarchy -version_kept 1 \
chip_top
Purging the cell: nand_macro ...
Purging the cell: chip_top ...
1
```

SEE ALSO

close_mw_cel(2)
copy_mw_cel(2)
create_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
rename_mw_cel(2)
save_mw_cel(2)

remove_net

Removes nets from the specified design.

SYNTAX

```
int remove_net net_list | -all | -all_empty_net | -all_single_net

net_list list
```

ARGUMENTS

net_list

Specifies a list of nets to be removed from the design. Each net name in *net_list* must exist in the design.

You must specify *net_list*, **-all**, **-all_empty_net** or **-all_single_net**. The *net_list*, **-all**, **-all_empty_net** and **-all_single_net** options are mutually exclusive.

-all

Removes all nets in the design. You must specify *net_list*, **-all**, **-all_empty_net** or **-all_single_net**. The *net_list*, **-all**, **-all_empty_net** and **-all_single_net** options are mutually exclusive.

-all_empty_net

Removes all empty nets in the design. You must specify *net_list*, **-all**, **-all_empty_net** or **-all_single_net**. The *net_list*, **-all**, **-all_empty_net** and **-all_single_net** options are mutually exclusive.

-all_single_net

Removes all single port or pin nets in the design. You must specify *net_list*, **-all**, **-all_empty_net** or **-all_single_net**. The *net_list*, **-all**, **-all_empty_net** and **-all_single_net** options are mutually exclusive.

DESCRIPTION

This command removes nets from the design. Net connections to pins or ports are disconnected.

To create nets, use the **create_net** command.

EXAMPLES

The following example removes all nets specified by the *N** collection in the current design.

```
prompt> get_nets "*"
{"NET0", "NET1", "NET2", "NET3", "MY_CHECK", "PARITY"}

prompt> remove_net "N*"
Removing net 'NET0' in design 'my_design'.
Removing net 'NET1' in design 'my_design'.
Removing net 'NET2' in design 'my_design'.
Removing net 'NET3' in design 'my_design'.

prompt> get_nets "*"
{"MY_CHECK", "PARITY"}
```

The following example removes all the remaining nets in the current design shown in the example given above.

```
prompt> remove_net -all
Removing net 'MY_CHECK' in design 'my_design'.
Removing net 'PARITY' in design 'my_design'.

prompt> get_nets "*"
{}
```

SEE ALSO

```
create_net(2)
current_design(2)
get_nets(2)
```

remove_net_shape

Removes net shapes.

SYNTAX

```
status_value remove_net_shape
   [-verbose]
   net_shapes

net_shapes list
```

ARGUMENTS

-verbose

Prints additional messages.

net_shapes

Specifies a nonempty collection of handles to net shapes.

DESCRIPTION

This command removes all specified net shapes. If the input collection is empty, the command returns **0** to indicate failure. Note that if there are some invalid handles in the input handle collection, the command line interpreter removes them from the collection and invokes the **remove_net_shape** command with the updated collection. This command returns **1** if successfully, otherwise it returns **0**.

EXAMPLES

The following example removes net shapes.

```
prompt> set a [get_net_shapes -of_objects n300]
{"VW6682", "HW7196", "VW6683"}


prompt> remove_net_shape $a
1
```

The following example removes net shapes specified by a pattern list.

```
prompt> remove_net_shape {hw7170 hw7171} -verbose
Removed net shape HW7170.
Removed net shape HW7171.
1
```

SEE ALSO

create_net_shape(2)
create_route_shape(2)
get_net_shapes(2)

remove_object

Removes a list of objects

SYNTAX

```
status remove_objects
   objects
```

ARGUMENTS

objects

Specifies the objects to be removed.

DESCRIPTION

Removes a list of objects from the current design.

EXAMPLES

The following example removes the selected objects:

```
Prompt > remove_objects [get_selection]
```

SEE ALSO

```
get_attribute(2)
get_designs(2)
```

remove_pin_guides

.prod icc Deletes the specified pin guides from the design. .prod syn Removes all pin guides from the design.

.prod all

SYNTAX

```
status remove_pin_guides
   -all | patterns
   [-verbose]
```

.prod icc

Data Types

```
   patterns     collection
```

.prod all

ARGUMENTS

-all

Removes all pin guides.

.prod icc

patterns

Specifies the collection of pin guides to delete. .prod all

-verbose

Enables verbose output.

.prod icc

If **-all** is specified, the total number of pin guides deleted is output; otherwise, the names of the deleted pin guides are output. .prod all

DESCRIPTION

This command deletes the pin guides from the design. .prod syn Design Compiler in topographical mode does not support the removal of pin guides by name. Use the **-all** option to remove all pin guides in the design, or use the following command:

```
prompt> remove_pin_guides *
```

.prod all

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example removes all pin guides:

```
prompt> remove_pin_guides -all
```

.prod icc

The following example removes selected pin guides:

```
prompt> remove_pin_guides [get_selection]
```

.prod all

SEE ALSO

```
get_pin_guides(2)
report_pin_guides(2)
```

remove_placement_blockage

Remove placement blockages.

SYNTAX

```
int remove_placement_blockage
   [-verbose]
   [-name name]
   patterns | -all


patterns  list
```

ARGUMENTS

-verbose

Prints more messages.

-name *name*

Specifies a name by which this command finds and removes a placement blockage.

Notice that the name should be a real name specified by you when creating the same placement blockage with the

```
create_placement_blockage -name
```

command, not a run time name, such as "pb7789". The "-name", "patterns" and "-all" are mutually exclusive.

patterns

Specifies the placement blockages to remove. The patterns can be a collection handle of blockage, or other formats, such as the following:

* or pb* Means all placement blockages. pb7789 Means one placement blockage whose object_id is 7789.

The "patterns", "-all" and "-name" are mutually exclusive.

-all

If specified, this command removes all placement blockages in the current design. The "-all", "patterns" and "-name" are mutually exclusive.

DESCRIPTION

This command removes all specified placement blockages.

EXAMPLES

The following example create a blockage for placement.

```
prompt> remove_placement_blockage -all
1

prompt> remove_placement_blockage [get_placement_blockages \
?  * -within {{2 2} {25 25}}]
1
```

SEE ALSO

```
create_placement_blockages(2)
create_route_guide(2)
get_placement_blockages(2)
get_route_guides(2)
remove_route_guide(2)
```

remove_port

Removes ports from the current design.

SYNTAX

```
status_value remove_port
   [-verbose]
   port_list

port_list list
```

ARGUMENTS

-verbose

Prints additional messages.

port_list

Specifies a list of ports to remove from the current design.

DESCRIPTION

This command removes a list of ports from the current design.

EXAMPLES

The following example removes all ports.

```
prompt> get_ports *
{"VDD", "VSS", "data2", "data1"}
```

The following example removes the port named *data2*.

```
prompt> remove_port data2
1
```

The following example removes the ports that start with *data*, specified by the **get_ports** command.

```
prompt> set a [get_ports data*]
{"data1"}
prompt> remove_port $a
1
```

SEE ALSO

```
create_port(2)
get_ports(2)
```

remove_route_guide

Removes route guides.

SYNTAX

```
status remove_route_guide
   [-verbose]
   patterns | -name name | -all
```

Data Types

```
   patterns list
   name     string
```

ARGUMENTS

-verbose

Prints additional messages.

patterns

Specifies the route guides to remove. You can specify pattern strings or collections of route guides. The pattern strings can include the wildcard characters "*" and "?".

The *patterns* argument, **-name** option, and **-all** option are mutually exclusive; you must specify one.

-name *name*

Specifies the name of the route guide to remove.

The name must be the user-specified name that was used when creating the route guide with the **create_route_guide -name** command, not a tool-generated name such as "rg7689".

The **-name** option, *patterns* argument, and **-all** option are mutually exclusive; you must specify one.

-all

Removes all route guides in the current design.

The **-all** option, *patterns* argument, and **-name** option are mutually exclusive; you must specify one.

DESCRIPTION

This command removes all specified route guides.

EXAMPLES

The following example removes the route guides specified by the **get_route_guides** command.

```
prompt> remove_route_guide \
   [get_route_guides -within {{2 2} {25 25}}]
1
```

The following examples remove all route guides.

```
prompt> remove_route_guide -all
1
prompt> remove_route_guide *
1
prompt> remove_route_guide rg*
1
```

The following example removes the route guide whose object ID is 7789.

```
prompt> remove_route_guide rg7789
1
```

SEE ALSO

```
create_placement_blockage(2)
create_route_guide(2)
get_placement_blockages(2)
get_route_guides(2)
remove_placement_blockage(2)
```

remove_routing_blockage

Removes the specified routing blockages.

SYNTAX

```
status remove_routing_blockage
   [-verbose]
   patterns
```

Data Types

```
   patterns     list
```

ARGUMENTS

-verbose

Prints additional messages.

patterns

Specifies the routing blockages. You can specify the patterns by using the following formats:

```
* A collection of routing blockages
* An asterisk (*), which indicates all routing blockages
* A string
  For example, RB_1234 matches the routing blockage named RB_1234
```

DESCRIPTION

This command removes the specified routing blockages.

EXAMPLES

The following example removes all routing blockages within the rectangle with the lower-left corner at {2 2} and the upper-right corner at {25 25}.

```
prompt> remove_routing_blockage \
[get_routing_blockages -within {{2 2} {25 25}}]
1
```

The following example removes all metal routing blockages.

```
prompt> remove_routing_blockage [get_routing_blockages -type metal]
1
```

SEE ALSO

create_routing_blockage(2)
get_routing_blockages(2)

remove_symbol_table

Removes the symbol table from the tool to save memory.

SYNTAX

```
int remove_symbol_table
```

ARGUMENTS

none

DESCRIPTION

This command removes the internal symbol table. A user generally never has to call this command. It is provided only to assist a user with large designs when memory is tight.

EXAMPLES

This example shows how to use this command.

```
prompt> remove_symbol_table
```

SEE ALSO

max_model_depth(2)

remove_terminal

Removes terminals.

SYNTAX

```
status_value  remove_terminal
   terminals
   [-verbose]

terminals   list
```

ARGUMENTS

terminals

Specifies patterns or a list of nonempty collection of handles to terminals.

-verbose

Prints more information.

DESCRIPTION

This command removes all specified terminals. If the input collection is empty, the command returns 0 to indicate failure.

If there are some invalid handles in the input handle collection, the command-line interpreter removes them from the collection and invokes the **remove_terminal** command with the updated collection.

EXAMPLES

The following example removes all terminals whose name match "clock*".

```
prompt> remove_terminal clock*
1
```

SEE ALSO

```
create_terminal(2)
get_terminals(2)
```

remove_text

Removes text.

SYNTAX

```
int remove_text text_list | -all
```

Data Types

text_list list

ARGUMENTS

text_list

Specifies the text to remove. The *text_list* value can include collections of text type, patterns that use wildcards, or certain command names. The *text_list* and **-all** options are mutually exclusive.

-all

Specifies to remove all text in the current design. The *text_list* and **-all** options are mutually exclusive.

DESCRIPTION

This command removes all specified text. Users can specify text by using the name of text or collections of text type which can be generated by commands like **get_text** and **get_selection**.

EXAMPLES

The following example shows the use of a pattern that removes the text object with object_id *6400*.

```
prompt> remove_text TEXT#6400
1
```

The following example shows the use of a wildcard pattern that removes all text.

```
prompt> remove_text *
1
```

The following example shows the use of a wildcard pattern that removes all text that start with *TEXT#*.

```
prompt> remove_text TEXT#*
1
```

The following example removes text specified by the **get_text** command.

```
prompt> remove_text [get_text \
? * -within {{10 9} {45 15}}]
1
```

Both of the following examples removes all text.

```
prompt> remove_text -all
1

prompt> remove_text
1
```

SEE ALSO

```
create_text(2)
get_texts(2)
```

remove_track

Removes tracks from the current design.

SYNTAX

```
status remove_track
   -all | patterns | -layer layer [-dir X | Y]
   [-verbose]
```

Data Types

```
   patterns     list
   layer        collection of one item
```

ARGUMENTS

-all

Removes all tracks in the current design.

The *patterns*, **-all**, and **-layer** arguments are mutually exclusive; you must specify one of these arguments.

patterns

Matches the track names in the current design against the specified patterns. You can specify the patterns by using the following formats:

- An asterisk (*), which indicates all tracks
- "TRACK_*", which indicates all tracks
- Track names, which are in the format TRACK_*object_id*

The *patterns*, **-all**, and **-layer** arguments are mutually exclusive; you must specify one of these arguments.

-layer *layer*

Specifies the routing layer from which to remove the track. You can specify only one layer, either by layer name, layer number, or a collection containing one layer.

The *patterns*, **-all**, and **-layer** arguments are mutually exclusive; you must specify one of these arguments.

-dir X | Y

Specifies the direction of the routing tracks to be removed. The valid values are **X** and **Y**.

By default, the direction is the routing direction of the layer specified in the physical library.

The option must be used with the **-layer** option.

-verbose

Prints additional messages.

DESCRIPTION

This command removes the specified tracks from the current design.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example removes all tracks that are inside {{2 2} {25 25}}.

```
prompt> remove_track [get_tracks -within {{2 2} {25 25}}]
1
```

The following example removes all tracks that are located on the metal2 layer.

```
prompt> remove_track [get_tracks -of_objects METAL2] -verbose
Removing track TRACK_4683
1
```

The following example removes the routing tracks from the routing layer named m3 on the floorplan.

```
prompt> remove_track -layer m3
Warning: Direction is not specified. Using the layer preferred direction.
(MWUI-125)
1
prompt> remove_track -layer m3 -dir X
1
```

SEE ALSO

create_track(2)
get_tracks(2)
report_track(2)

remove_user_shape

.prod icc Removes objects that are user shapes. You can specify objects by a collection or by name. .prod syn Removes objects that are user shapes. .prod all

SYNTAX

```
status remove_user_shape
   [-verbose]
   user_shapes
```

Data Types

```
   user_shapes     collection
```

ARGUMENTS

-verbose

Prints additional information. If this option is not specified, the tool suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

user_shapes

.prod icc Specifies the user shapes to remove. .prod syn Specifies an asterisk (*) as the pattern, and all user shapes are removed. .prod all

DESCRIPTION

This command removes user shapes. It returns 1 if successful or 0 if it fails.

.prod icc

If you specify an invalid collection, the command-line interpreter removes it and runs the **remove_user_shape** command with the updated collection.

.prod all

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example removes user shapes.

```
.prod icc
prompt> set a [get_user_shape]
{"RECTANGLE#6682" "TRAPEZOID#7196" "POLYGON#6683"}
prompt> remove_user_shape $a
1
.prod syn
prompt> remove_user_shape *
1
.prod all
```

SEE ALSO

```
create_user_shape(2)
get_user_shapes(2)
remove_net_shape(2)
```

remove_via

Removes vias.

SYNTAX

```
status_value remove_via
   [-verbose]
   vias
```

Data Types

```
   vias  list
```

ARGUMENTS

-verbose

Prints additional messages.

vias

Specifies a nonempty collection of handles to vias.

DESCRIPTION

This command removes all specified vias. If the input collection is empty, the command returns 0 to indicate failure. If any invalid handles occur in the input handle collection, the command line interpreter removes them from the collection and invokes the **remove_via** command with the updated collection.

This command returns 1 if successful, or 0 if it fails.

EXAMPLES

The following example removes the vias specified by the **get_vias** command:

```
prompt> remove_via [get_vias -of_objects n300]
1
```

SEE ALSO

```
create_net_shape(2)
create_via(2)
get_net_shapes(2)
get_vias(2)
remove_net_shape(2)
```

remove_via_master

Removes via masters from the current design.

SYNTAX

```
status remove_via_master
   -all | via_master_name
```

Data Types

```
   via_master_name list
```

ARGUMENTS

-all

Removes all via masters from the current design. This argument and *via_master_name* are mutually exclusive.

via_master_name

Specifies the via masters to be removed. You can give a via master name list or a via master name collection as the argument. This argument and **-all** are mutually exclusive.

DESCRIPTION

This command removes specified via masters from current design.

The via master can be removed only if there are no instances still referring to it.

Only via masters created with the **create_via_master** command can be removed. A via defined in the technology file using the ContactCode syntax cannot be removed by this command.

The command returns 1 if successful or 0 if it fails.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following command removes a via master naming 'viamaster1' from current design:

```
prompt> remove_via_master viamaster1
```

The following command removes two via masters by specifying via master name list:

```
prompt> remove_via_master {viamaster1 viamaster2}
```

The following command removes via masters by specifying a via master collection:

```
prompt> remove_via_master [get_via_master viamasterName1*]
```

The following command removes all the via masters from current design by using the option '**-all**':

```
prompt> remove_via_master -all
```

SEE ALSO

```
create_via_master(2)
get_via_masters(2)
```

remove_voltage_area

Removes voltage areas.

SYNTAX

```
int remove_voltage_area
   patterns | -all
```

Data Types

```
   patterns  list
```

ARGUMENTS

patterns

Matches voltage area names against patterns in the current design. The patterns can include the wildcard characters ***** (asterisk) and **?** (question mark). This option can include collections of voltage area type.

The *patterns* and **-all** arguments are mutually exclusive.

-all

Removes all voltage areas in the current design. The *patterns* and **-all** arguments are mutually exclusive.

DESCRIPTION

This command removes all specified voltage areas.

EXAMPLES

The following example removes the voltage areas specified by the **get_voltage_areas** command.

```
prompt> remove_voltage_area [get_voltage_areas \
? -of_objects buffdaG1B2I1_1]
1
```

The following example removes all voltage areas.

```
prompt> remove_voltage_area -all
1
```

SEE ALSO

```
create_voltage_area(2)
get_voltage_areas(2)
```

rename_mw_cel

Renames a Milkyway cel.

SYNTAX

```
int rename_mw_cel
   old_name new_name
   [-all_version]
```

Data Types

```
   old_name  string
   new_name  string
```

ARGUMENTS

old_name

Specifies the name of the Milkyway cel to be renamed.

new_name

Specifies the new name of the Milkyway cel. The *new_name* should not match the name of any cell masters in the Milkyway cel specified by the *old_name*.

-all_version

Rename the design name for all versions.

DESCRIPTION

This command renames a Milkyway cel.

By default, this command renames all versions of the Milkyway cel specified by *old_name*. After you clear a variable by typing the following in the command line, this command only renames the newest version of that Milkyway cel:

```
set mw_rename_mw_cel_all_versions 0
```

Each Milkyway cel name must be unique within the library containing that Milkyway cel. An error occurs if the *new_name* you specify is already used in the library.

EXAMPLES

The following example renames the *test* Milkyway cel to *bill*:

```
prompt> get_mw_cels *
{"top", "test"}

prompt> rename_mw_cel test bill
1

prompt> get_mw_cels *
{"top", "bill"}
```

SEE ALSO

close_mw_cel(2)
copy_mw_cel(2)
create_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
remove_mw_cel(2)

rename_mw_lib

Renames a Milkyway library.

SYNTAX

```
status_value rename_mw_lib
   -from lib_name
   -to lib_name
```

ARGUMENTS

-from *lib_name*

Specifies the name of the Milkyway library that will be renamed. The specified Milkyway library must not be open in current session.

-to *lib_name*

Specifies the new name of the Milkyway library.

DESCRIPTION

This command renames a Milkyway library.

If this library is a reference library of another main library, you may need to use the **add_reference_library** and **remove_reference_library** commands to update reference library information in the main library.

A status indicating success or failure is returned.

EXAMPLES

The following example changes the name of the library from access05 to access06

```
prompt> rename_mw_lib -from access05 -to access06
```

SEE ALSO

copy_mw_lib(2)

replace_cell

Replaces a list of cells' reference design.

SYNTAX

```
status_value replace_cell
   [-verbose]
   -design new_design
   cell_list

cell_list   list
new_design  list
```

ARGUMENTS

-verbose

Prints more messages.

-design *new_design*

Specifies the name of the new design by which the reference design of the cells is to be replaced.

cell_list

Specifies a list of cells to replace.

DESCRIPTION

This command replaces the cell in the current design. Connections to the cell are maintained if the port names of the new reference design match the port names of the existing cell reference design. Ports in the old reference design that have no equivalent port in the new reference design are disconnected. Ports in the new reference design that have no equivalent port in the old reference design are left unconnected.

EXAMPLES

The following example replaces specified cells.

```
prompt> get_cells or2t*
{"or2t_1", "or2t_2", "or2t_3"}

prompt> get_attri [get_cells or2t_1] ref_name
OR2T

prompt> replace_cell -design OR3T or2t*
1

prompt> get_attri [get_cells or2t_1] ref_name
OR3T
```

SEE ALSO

create_cell(2)
get_cells(2)
remove_cell(2)

replace_tlu_plus_file

Replaces the TLU+ file for a Milkyway library.

SYNTAX

```
status_value
replace_tlu_plus_file
   [-library mw_libs | -lib_id lib_id]
   tlu_plus_file
```

Data Types

```
mw_libs        string
lib_id         string
tlu_plus_file  string
```

ARGUMENTS

-library *mw_libs*

Specifies the Milkyway libraries whose TLU+ file is to be replaced. The *mw_libs* value can be one or more library names or a list of a collection of libraries. By default, the command uses the current Milkyway library. The **-lib_id** and **-library** options are mutually exclusive.

-lib_id *lib_id*

Specifies the ID of the Milkyway library whose TLU+ file is to be replaced. By default, the command uses the current Milkyway library. The **-lib_id** and **-library** options are mutually exclusive.

tlu_plus_file

Specifies the tlu_plus_file to replace the existing one in the Milkyway library.

DESCRIPTION

This command replaces the TLU+ file for a Milkyway library. It returns a status indicating success or

failure.

EXAMPLES

The following example replaces the TLU+ information of the current Milkyway library with *new_tlu_plus_file* information.

```
prompt> replace_tlu_plus_file new_tlu_plus_file
1
```

SEE ALSO

dump_tlu_plus_file(2)

report_antenna_rules

Reports all of the antenna rules stored in the library.

SYNTAX

```
status_value report_antenna_rules
   [mw_lib | -lib lib_id]
   [-output file_name]

mw_lib list


lib_id string


file_name string
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be updated. The value of *mw_lib* can be a library name or a one-element collection of a library. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-lib *lib_id*

Specifies the ID of the Milkyway library to be updated. The *mw_lib* and **-lib** options are mutually exclusive. The default is to use the current Milkyway library.

-output *file_name*

The name of a specified output file.

DESCRIPTION

This command reports all of the antenna rules stored in the library to a file or to the window. The command returns a status indicating success or failure.

EXAMPLES

```
prompt> report_antenna_rules
```

SEE ALSO

```
define_antenna_layer_ratio_scale(2)
define_antenna_layer_rule(2)
define_antenna_rule(2)
remove_antenna_rules(2)
```

report_attribute

Reports the attributes on one or more objects.

SYNTAX

```
string report_attribute
   [-application]
   [-class class_name]
   object_list
```

Data Types

```
   class_name   string
   object_list  list
```

ARGUMENTS

-application

Lists application attributes and user-defined attributes.

-class *class_name*

Specifies the class name for the object specified in *object_list*, if the element of *object_list* is a name. The valid values for *class_name* are the following:

design
port
cell
pin
net
lib
lib_cell
lib_pin
clock
bound
net_shape
placement_blockage
route_guide
route_shape
terminal
text

object_list

Reports a list of objects. Each element in the list is either a collection or a pattern that is combined with the *class_name* to find the objects.

DESCRIPTION

This command generates a report of attributes on the specified objects. By default, only user-defined attributes are displayed.

EXAMPLES

The following example reports attributes on a specified net object.

```
prompt> report_attribute [get_nets clk2] -app

****************************************
Report : Attribute
LIBRARY: astro_tcl
DESIGN : top
Date   : Wed Sep  8 10:26:17 2004
****************************************


Design                    Object     Type      Attribute Name        Value
--------------------------------------------------------------------------
top                       clk2       string    base_name             clk2
top                       clk2       boolean   dont_touch            false
top                       clk2       boolean   ideal_net             false
top                       clk2       boolean   is_physical           false
top                       clk2       boolean   is_tie_high_net       false
top                       clk2       boolean   is_tie_low_net        false
top                       clk2       string    net_type              Clock
top                       clk2       int       num_overall_pins      2
top                       clk2       int       num_pins              2
top                       clk2       string    owner                 top
top                       clk2       string    route_length  {{METAL3 28.000}\
                                                              {METAL2 8.110}}
top                       clk2       int       cell_id               3
top                       clk2       string    full_name             clk2
top                       clk2       string    mw_name               clk2
top                       clk2       string    name                  clk2
top                       clk2       int       number_of_wires       8
top                       clk2       string    object_class          net
top                       clk2       int       object_id             2816
--------------------------------------------------------------------------
```

The following example limits the reporting only to user-defined attributes.

```
prompt> define_user_attribute -class net attr_s -type string
Info:User-defined attribute 'attr_s' on class 'net'.
```

```
1
prompt> set_attribute -class net clk2 attr_s hello
Info:Setting attribute 'attr_s' on net 'clk2'.
{"clk2"}
prompt> report_attribute -class net clk2

****************************************
Report : Attribute
LIBRARY: astro_tcl
DESIGN : top
Date   : Wed Sep  8 10:41:31 2004
****************************************


Design                    Object     Type      Attribute Name        Value
--------------------------------------------------------------------------------
top                       clk2       string    attr_s                hello
--------------------------------------------------------------------------------
```

SEE ALSO

define_user_attribute(2)
get_attribute(2)
remove_user_attribute(2)
report_attribute(2)
set_user_attribute(2)

report_check_library_options

Reports the option values set by the set_check_library_physical_options command for check_library.

SYNTAX

```
int report_check_library_options
   [-physical]
   [-default]
```

ARGUMENTS

-physical

Reports the values or status for the physical library checking options set by set_check_library_physical_options.

-default

Reports the default values for the library checking options.

DESCRIPTION

The **report_check_library_options** command reports the option values for checking the physical library that are set with set_check_library_physical_options for **check_library**.
The report_check_library_options command is used after running set_check_library_physical_options.

EXAMPLES

```
The following example checks all data for the specified library test.mw, and
reports the option values.

prompt> set_check_library_physical_options -all
prompt> report_check_library_options -physical
1

Report check library options
**********************************
Check pin routeability                  : true
Check missing and mismatched views      : true
Check antenna rule                      : true
Check signalEM rule                     : true
Report same name cell                   : true
Report rectilinear cell                 : true
Report physical property                : place route
Report physical only cell               : true
Check tech consistency                  : true
Check technology data                   : true
Check cell DRC                          : true
Check macro metal density               : true
Cell name                               : (null)
Check all                               : true
**********************************
1
```

SEE ALSO

check_libray(2)
set_check_library_physical_options(2)

report_collections

Reports all of the collections currently present in the tool.

SYNTAX

```
<status> report_collections
```

ARGUMENTS

None.

DESCRIPTION

The **report_collections** command reports report the set of active collections, and the number of elements in each.

Since there may be some memory overhead associated with collections, it is useful to know what collections are present.

EXAMPLES

The following is an example of **report_collections**.

```
prompt> report_collections
```

SEE ALSO

collections(2)
filter_collection(2)
query_objects(2)
remove_collections(2)

report_milkyway_version

Reports information for the specified cell, including the Milkyway data model version and revision information.

SYNTAX

```
status report_milkyway_version
   [-cell cell_name | -all]
```

Data Types

```
   cell_name         string
```

ARGUMENTS

-cell *cell_name*

Generates a report for the specified cell.

-all

Generates a report for all cells in the current Milkyway design library.

DESCRIPTION

This command reports the following for each reported cell:

```
 - The cell data model version
 - The product version that is compatible with the cell data model version
 - The product release version that created the initial design
 - The cell's initial creation time
 - The product release version that modified the design last
 - The cell's last modification time
```

All cells must be closed when running this command.

Either the -cell or -all option must be specified. If neither option is specified, this command will display an error message and fail. If both options are specified, the -all option will take precedence and a report is generated for all cells.

EXAMPLES

The following example displays a report for the cell "top" in the current Milkyway design library.

```
prompt> open_mw_lib design
prompt> report_milkyway_version -cell top

Cell Name               top.CEL;2
Data Model              1.2
Compatible Product      A-2007.12 And Older Versions
Creator                 Z-2007.03-ICC-SP2-1
Creation Time           Fri Jul 27 11:10:04 2007
Modifier                Z-2007.03-ICC-SP2-1
Modification Time       Mon Jul 30 17:34:34 2007
1
```

The following example displays a report for all cells in the current Milkyway design library.

```
prompt> open_mw_lib design
prompt> report_milkyway_version -all

Cell Name               top.CEL;1
Data Model              1.2
Compatible Product      A-2007.12 And Older Versions
Creator                 Z-2007.03-ICC-SP2-1
Creation Time           Mon Jul 23 16:00:45 2007
Modifier                Z-2007.03-ICC-SP2-1
Modification Time       Mon Jul 23 16:01:08 2007

Cell Name               top.CEL;2
Data Model              1.2
Compatible Product      A-2007.12 And Older Versions
Creator                 Z-2007.03-ICC-SP2-1
Creation Time           Fri Jul 27 11:10:04 2007
Modifier                Z-2007.03-ICC-SP2-1
Modification Time       Mon Jul 30 17:34:34 2007

Cell Name               macro1.CEL;1
Data Model              1.2
Compatible Product      A-2007.12 And Older Versions
Creator                 Z-2007.03-ICC-SP2-1
Creation Time           Mon Jul 23 16:00:46 2007
Modifier                Z-2007.03-ICC-SP2-1
Modification Time       Mon Jul 23 16:01:08 2007

Cell Name               macro1.CEL;2
Data Model              3.2
Compatible Product      B-2008.09
Creator                 Z-2007.03-ICC-SP2-1
Creation Time           Mon Jul 23 16:00:46 2007
Modifier                B-2008.09-ICC-SP-INTERNAL
Modification Time       Tue Sep  9 14:30:54 2008
1
```

SEE ALSO

convert_mw_lib(2)
open_mw_lib(2)

report_mw_cel

Displays information about the specified designs.

SYNTAX

```
status_value report_mw_cel
   [design_list]
   [-quiet]

design_list   list
```

ARGUMENTS

design_list

Specifies the designs to report.

You can specify designs by name, name pattern, or the design collection's name. For example, *top* matches a design named *top* in the current library. Specifying *top** matches all designs whose names begin *top*. The command **report_mw_cel [get_mw_cel *]** reports all designs in the current library.

If not specified, *current design* will be used.

-quiet

Turns off warning messages.

DESCRIPTION

This command lists information about the contents of the design.

EXAMPLES

The following example reports all of the designs in the current library:

```
prompt> report_mw_cel *
```

SEE ALSO

close_mw_cel(2)
copy_mw_cel(2)
create_mw_cel(2)
current_mw_cel(2)
get_mw_cels(2)
open_mw_cel(2)
remove_mw_cel(2)
rename_mw_cel(2)
update_mw_cel(2)

report_mw_lib

Displays information about a Milkyway library.

SYNTAX

```
status_value report_mw_lib
   [-unit_range]
   [-mw_reference_library]
   [mw_lib]
```

ARGUMENTS

mw_lib

Specifies the Milkyway library to be reported. If no mw_lib specified, the current Milkyway library is used.

-unit_range

Indicates to list the information of the unit. If both are omitted, only technology information is reported.

-mw_reference_library

Prints the list of reference libraries for the Milkyway library. If the library is not be specified, the current Milkyway library is used.

DESCRIPTION

Displays information about a Milkyway library.

The **-unit_range** option reports unit names, parasitic and physical attributes, and the ranges of their values associated with the Milkyway library.

A status indicating success or failure is returned.

EXAMPLES

The following example displays unit information about a Milkyway library.

```
prompt> report_mw_lib design -unit_range

Library: design
Tech. Attr.   Unit       Resolution   Min. Value    Max. Value
===========   ========   ==========   ==========   ===========
length        micron           1000        0.001   2147483.647
time          ns               1000        0.001   2147483.647
capacitance   pf           10000000    0.0000001   214.7483647
resistance    kohm         10000000    0.0000001   214.7483647
inductance    nh                100         0.01   21474836.47
current       mA               1000        0.001   2147483.647
voltage       V              100000      0.00001   21474.83647
power         pw               1000        0.001   2147483.647

Layer: POLY
Mask name: poly
Attribute                 Minimum    Maximum
======================   ========   ========
thickness                 0.0e+00    0.0e+00
unit resistance           0.0e+00    0.0e+00
unit capacitance          0.0e+00    0.0e+00
unit channel cap..        0.0e+00    0.0e+00
unit sidewall cap.        0.0e+00    0.0e+00
unit channel side cap.    0.0e+00    0.0e+00
unit inductance           0.0e+00    0.0e+00
height from substrate     0.0e+00    0.0e+00
delta width               0.0e+00    0.0e+00
min. area dimension       0.0e+00
min. object width         1.8e-01
min. object spacing       2.5e-01
max. wire  length                    0.0e+00
max. RC seg. length                  0.0e+00
max. current density                 0.0e+00
intracap. dist. ratio                3.0e-03

 ......

1
```

SEE ALSO

close_mw_lib(2)
open_mw_lib(2)

report_track

Reports the routing tracks for a specified layer or for all layers.

SYNTAX

```
int report_track
   [-layer layer]
   [-dir X | Y]
```

Data Types

```
   layer     string
```

ARGUMENTS

-layer *layer*

Specifies the routing layer to use the routing tracks. You can use layer name, layer number, or a collection containing one layer object.

The default is to report the routing tracks on all layers.

-dir X | Y

Specifies the direction how routing tracks are placed. The valid values are **X** and **Y**; specify either. The default is to report routing tracks in both direction.

DESCRIPTION

This command reports a group of routing tracks for the routing layer.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example reports routing tracks on the floorplan.

```
prompt> report_track
****************************************
Report track
Design : design
Version: Y-2006.06-ICC-SP2
Date   : Thu Jul  6 00:34:03 2006
****************************************
Layer          Direction     Start         Tracks    Pitch          Attr
--------------------------------------------------------------------------
Attributes :
         usr : User defined
         rt  : Route66 defined
         def : DEF defined

m2                Y         233.720        2861      0.560
m2                X         233.720        2649      0.560
m3                X         233.720        2649      0.560
m3                Y         233.720        2861      0.560
m2                Y         233.720        2861      0.560
m4                Y         233.720        2861      0.560
m4                X         233.720        2649      0.560
m3                X         233.720        2649      0.560
m5                X         233.720        2649      0.560
m5                Y         233.720        2861      0.560
m4                Y         233.720        2861      0.560
m6                Y         233.720        2861      0.560
m6                X         232.600        1327      1.120
m5                X         232.600        1327      1.120
m1                X         20.000         10        2.100          usr
1
prompt> report_track -layer m3 -dir horizontal
****************************************
Report track
Design : design
Version: Y-2006.06-ICC-SP2
Date   : Thu Jul  6 00:36:25 2006
****************************************
Layer          Direction     Start         Tracks    Pitch          Attr
--------------------------------------------------------------------------
Attributes :
         usr : User defined
         rt  : Route66 defined
         def : DEF defined

m3                Y         233.720        2861      0.560
1
```

SEE ALSO

```
create_track(2)
remove_track(2)
```

report_voltage_area

Reports the voltage areas in the design. .prod syn This command is supported only in topographical mode. .prod all

SYNTAX

```
status report_voltage_area
   [-connection]
   [-operating_condition]
   .prod syn
   [-name list]
   [-nosplit]
   [-verbose]
   .prod all
   -all | patterns
```

Data Types

```
   list        list
   patterns         list
```

ARGUMENTS

-connection

Reports pin connections between voltage areas. By default, pin connections are not reported.

-operating_condition

Reports operating conditions. By default, operating conditions are not reported.

-all

Reports all voltage areas in the design. Voltage areas are created by using the **create_voltage_area** command for the hierarchical blocks.

This option is mutually exclusive with the *patterns* argument.

patterns

.prod syn Specifies the list of hierarchical cells that each represent a voltage area.

.prod icc

Specifies the voltage areas to be reported. The patterns can be a collection handle of voltages or names of patterns. You can use the *get_voltage_areas* command to specify voltage areas to be reported.

.prod all

This argument is mutually exclusive with the **-all** option.

.prod syn

-name *list*

Specifies the names of voltage areas to be reported.

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

-verbose

Reports both maximum and minimum operating conditions. .prod all

DESCRIPTION

This command reports the user-specified voltage area constraints in the design, as specified by the **create_voltage_area** command. If you use the **-all** option, all voltage areas in the design are reported.

The report includes voltage area name, hierarchical blocks associated with voltage area, voltage area geometry, area of the voltage area, and utilization for the voltage area. The area is reported in square microns.

Voltage areas can physically be completely nested, that is, one voltage area lies completely inside another voltage area. When considering the area or utilization of the outside voltage area, the area of the inner voltage area is excluded.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example reports the voltage area va1:

```
.prod icc
prompt> report_voltage_area va1
.prod syn
prompt> report_voltage_area -name va1
.prod all
****************************************
Voltage area name va1
Block(s) :PPL_LBUS_CORE
Voltage area geometry : { 19.87 29.86 576.47 1148.89 }
Voltage area region Area : 622854.88
Voltage area Utilization : 0.66
Voltage Area Guard-band (X, Y) : (1, 1)
****************************************
1
```

SEE ALSO

```
create_voltage_area(2)
.prod icc
get_voltage_areas(2)
update_voltage_area(2)
.prod all
remove_voltage_area(2)
```

resize_polygon

Returns a list of polygons whose edges have been pushed outwards or inwards (away from the area covered by the target polygon) by a specified distance.

SYNTAX

```
list resize_polygon
   polygon
   -size size
```

Data Types

```
   polygon       list
   size          double
```

ARGUMENTS

polygon

One polygon which is represented as a list of points. The format for a polygon is: {{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}. Besides, a list of one polygon is also supported as input for this option, with the format: {{{x1 y1} {x2 y2} ... {xN yN} {x1 y1}}}.Pay attention that the valid polygon is rectilinear, so the adjacent points have one same coordinate. The coordinate unit is specified in technology file (usually it is micron).

-size *size*

Specified double value used to adjust the edges of the polygon

DESCRIPTION

This command returns a list of polygons whose edges have been pushed outwards or inwards (away from the area covered by the target polygon) by a specified size. If the size is *positive*, the polygon will be oversized in which case the edges are pushed outwards and any gaps less than 2*size units will be filled;

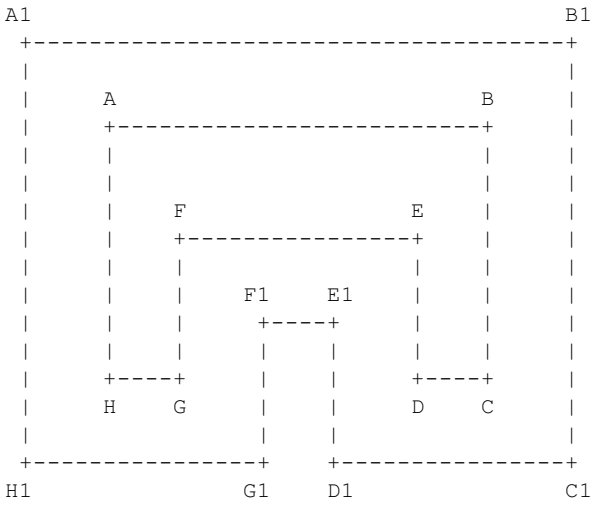
otherwise the polygon will be undersized in which case the edges are pushed inwards. Each returned polygon will be represented as points list.

It is important to note that this command may return a list of more than one polygon which represents a disjoined rectilinear region if the size value is *negative*. So do not directly pass the result of this command as a parameter to another polygon command when negative size is specified. Tcl list command like **foreach**, **lindex** can be used to extract each polygon from the returned list, and then pass each polygon to other polygon command. But if the size value is positive, this command will return only one polygon, and then the result can be passed directly to other polygon commands, including **convert_from_polygons**, and **get_polygon_area**, **resize_polygon** and **compute_polygons**.

Before this command is used, the library should be opened.

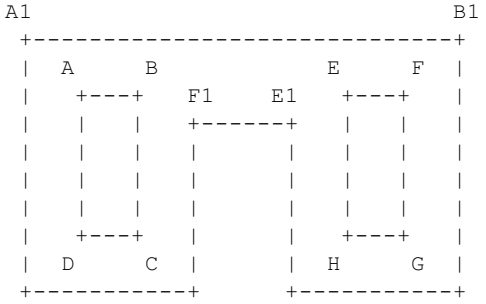
EXAMPLES

The following example returns the polygon A1-B1-C1-D1-E1-F1-G1-H1-A1 which is oversized from polygon A-B-C-D-E-F-G-H-A by size 10.




```
prompt> resize_polygon {{10 40} {60 40} {60 10} {50 10} {50 30} \
{20 30} {20 10} {10 10} {10 40}} -size 10
{{70 50} {70 0} {40} {40 20} {30 20} {30 0} {0 0} {0 50} {70 50}}
```

The following example returns two result polygons A-B-C-D-A and E-F-G-H-E after the polygon A1-B1-C1-D1-E1-F1-G1-H1-A1 is undersized by negative size 10.



```
 H1           G1    D1           C1
```

```
prompt> resize_polygon {{0 45} {80 45} {80 0} {50 0} {50 30} {30 30} \
{30 0} {0 0} {0 45}} -size -10
{{10 35} {20 35} {20 10} {10 10} {10 35}} {{60 35} {70 35} {70 10}
{60 10} {60 35}}
```

The following example passed the result of resize_polygon as a parameter to resize_polygon again.

```
prompt> resize_polygon -size -10 [resize_polygon {{10 40} {50 40} {50 10} \
{40 10} {40 30} {20 30} {20 10} {10 10} {10 40}} -size 10]
{10 40} {50 40} {50 10} {10 10} {10 40}
```

SEE ALSO

convert_to_polygon(2)
convert_from_polygon(2)
compute_polygons(2)

restore_design_settings

Restores the session variable settings previously saved with the **save_design_settings** command.

SYNTAX

```
status restore_design_settings
   -library | -design
```

ARGUMENTS

-library

Restores design settings that were previously saved in the current library.

-design

Restores design settings that were previously saved in the current design cell.

DESCRIPTION

This command restores the design settings that were previously saved with the **save_design_settings** command. It applies the restored settings to the current session. Before you restore the saved design settings, you can use the **write_design_settings** command to see which variables will be overwritten and their new values.

EXAMPLES

The following example restores the design settings saved in the current library.

```
prompt> restore_design_settings -library
```

```
1
```

SEE ALSO

save_design_settings(2)
write_design_settings(2)

rotate_objects

Rotate one or more cells, ports, terminals, net_shapes, or route_shapes.

SYNTAX

```
status_value rotate_objects
   -to orientation_value | -by rotation_value
   [-pivot_point point | -fixed_ll]
   objects
```

ARGUMENTS

-to *orientation_value*

Specifies the new orientation of the object by using either DEF or Floorplan Compiler notation. See the **orientation_value** man page for more information about specifying orientation values. Only cells can be rotated by this option. The **-to** option and the **-by** option are mutually exclusive. You must include one of these options.

-by *rotation_value*

Specifies the change in orientation of the object by using the following keywords for rotation_value:

```
CW90
CW180
CW270
CCW90
CCW180
CCW270
FLIPX
FLIPY
```

The **-to** option and the **-by** option are mutually exclusive. You must specify one of these options.

-pivot_point *point*

Specifies the point for pivoting. The **-pivot_point** option and the **-fixed_ll** option are mutually exclusive. You can only specify one of them. If both are omitted, the default uses the lower-left corner of the bounding box of the object or collection of objects.

-fixed_ll

Fixes the lower-left corner of the bounding box of the object or collection of objects during rotation. This is also the default behavior. The **-fixed_ll** option and the **-pivot_point** option are mutually exclusive. You can only specify one of them.

objects

Specifies the cells, ports, port shapes, net shapes, keepouts or bounds to be rotated.

DESCRIPTION

This command rotates the specified objects around a pivot point by the specified absolute (**-to** option) or delta (**-by** option) orientation.

EXAMPLES

The following example rotates cell nand23 by 90 degrees clockwise.

```
prompt> rotate_objects -by CW90 [get_cells nand23]
1
```

SEE ALSO

move_objects(2)
remove_objects(2)

save_design_settings

Saves the session variable settings into the current library or design cell.

SYNTAX

```
status save_design_settings
   -library | -design
   [-input file]
```

Data Types

```
   file              string
```

ARGUMENTS

-library

Saves the design settings (modified variable settings) into the current library.

-design

Saves the design settings into the current design cell. To preserve these settings, you need to save the design to disk using the **save_mw_cel** command.

-input *file*

Saves the design settings contained in the given input file into the library or design cell, instead of saving the current design settings.

DESCRIPTION

This command saves the design settings (variable settings of the session) into the current library or design cell, depending on whether you use the **-library** or **-design** option. Then, in a later session, you can restore the variable settings by using the **restore_design_settings** command.

By default, this command saves only the session's application variables that have been changed from their default values, like the default behavior of the **write_app_var** command.

If you specify an input file with the **-input** option, the commands saves contents of this file (instead of the current session settings) into the library or design. Saving these design settings does not apply them to the current session.

To report the design settings that have been saved, use the **write_design_settings** command.

If design settings already exist in the current library or design cell, the new design settings overwrite the existing ones.

EXAMPLES

The following example saves the design settings into the current library.

```
prompt> save_design_settings -library
1
```

SEE ALSO

```
write_app_var(2)
write_design_settings(2)
restore_design_settings(2)
```

save_mw_cel

Saves the current Milkyway design.

SYNTAX

```
status_value save_mw_cel
   [-as string]
   [-increase_version]
   [-overwrite]
   [design_list]
```

Data Types

```
   string  string
   design_list       string
```

ARGUMENTS

-as *string*

Saves the Milkyway design as a new Milkyway design with the given name.

-increase_version

Increases the version of the Milkyway design to be saved. If this option is not specified, it will overwrite current version of specified Milkyway design. If used with **-as** option, it means to increase the version of Milkyway design specified in as option.

-overwrite

Overwrites the existing Milkyway design.

design_list

Specifies the Milkyway designs to be saved.

DESCRIPTION

This command saves the current Milkyway design or specified Milkyway designs. By default, it will overwrite current version of the Milkyway design.

You should not use a "." in the CEL name because it is considered a special character to specify a {name, viewname} combination. If a "." is specified in the design name, the substring before the "." is treated as the name and the substring after the "." is as a view name. For example, if the name specified is "abc.def", then the design name will be "abc" and is created in the view "def". If more than one "." exists in the name specified, then the substring after the last "." is considered the view name while all of the substring preceding the ".", including any "." is considered the design name.

EXAMPLES

The following example saves the current Milkyway design as *my_new_cel*:

```
prompt> save_mw_cel -as my_new_cel
1
```

SEE ALSO

close_mw_cel(2)
create_mw_cel(2)
open_mw_cel(2)

scheme

Interprets a single Scheme command.

SYNTAX

scheme {*scheme_command*}

scheme_command list

DESCRIPTION

This command allows you to evaluate a single scheme command. If you have multiple scheme commands to process, use the **begin_scheme** command instead of **scheme**.

Since most scheme commands require "" (set of double quotation marks) around each argument and have other syntax differences with Tcl, it is generally necessary to enclose the *scheme_command* in {} (braces).

EXAMPLES

The following example executes a single scheme command.

```
prompt> scheme {functions "ata"}
```

SEE ALSO

begin_scheme(2)

set_attached_file_link_mode

Sets the link mode for the design's attached file.

SYNTAX

```
status set_attached_file_link_mode [mode]
```

Data Types

```
mode  int
```

ARGUMENTS

mode

Indicates the link mode for the design's attached file. The default is to return the current value of the link mode.

DESCRIPTION

This command checks the current value of the link mode or sets the link mode. If the link mode is **TRUE** when a design is copied, its attached files are linked, not copied. If the mode is **FALSE**, they are copied. By default, this link mode is set to **FALSE**. By setting the link mode to **TRUE**, disk space for duplication of attached files can be saved. See the man page for the **purge_attached_file** command for more information.

There are four Scheme commands that may be impacted by this link mode, too. They are

```
dbArchiveCellToLib
dbRestoreHierDesign
dbArchiveDesign
dbArchiveHierDesign
```

If the link mode is set to **TRUE**, the linkage among the design's attached files in the source library are limitedly preserved in the target library. For example, if a library named *my_lib* has four attached files that

share one physical file (they are hard-linked), and the files are:

```
CEL/top:1_1
    top:1_2
    top:2_1
    new:1_1
```

Then you issue the following commands, one by one:

```
dbArchiveDesign "my_lib" "top" "arch_lib" 0
dbArchiveDesign "my_lib" "new" "arch_lib" 0
```

The result (in the archive library *arch_lib*) is the following:

1. *top:1_1*, *top:1_2*, and *top:2_1* are still linked together.
2. *new:1_1* is a separate file.

EXAMPLES

The following example checks the current value of the link mode.

```
prompt> set_attached_file_link_mode
0
```

The following example sets the link mode. This enables linking of the design's attached file when a design is copied.

```
prompt> set_attached_file_link_mode 1
1
```

SEE ALSO

purge_attached_file(2)

set_attribute

Sets an attribute to a specified value on the specified list of objects.

SYNTAX

```
collection set_attribute
   [-class class_name]
   object_list
   attribute_name
   attribute_value
   [-quiet]

class_name string
object_list list
attribute_name string
attribute_value string
```

ARGUMENTS

-class *class_name*

Specifies the class name for the object specified in *object_list*, if the element of *object_list* is a name. The following is a list of valid values for *class_name*:

design
port
cell
pin
net
lib
lib_cell
lib_pin
clock
bound
net_shape
placement_blockage
route_guide
route_shape
terminal
text

object_list

A list of objects on which the attribute is to be set. Each element in the list is either a collection or a pattern that is combined with the *class_name* to find the objects.

attribute_name

Specifies the name of the attribute to be set.

attribute_value

Specifies the value of the attribute. The datatype must be the same as that of the attribute.

-quiet

Turns off the warning message that would otherwise be issued if the attribute or objects are not found.

DESCRIPTION

This command sets the value of an attribute on an object. For a complete listing of attributes, refer to the **attributes** man page.

This command creates a collection of objects that have the specified attribute value set. A returned empty string indicates that no object has been set.

EXAMPLES

The following example defines an attribute named *X* for cells, then sets the value on all cells in this level of the hierarchy.

```
prompt> define_user_attribute -type int -class cell X
cell
prompt> set_attribute [get_cells *] X 30
{"U1"}
```

SEE ALSO

```
collections(2)
define_user_attribute(2)
get_attribute(2)
list_attributes(2)
remove_attribute(2)
```

set_check_library_physical_options

Sets the options used by the **check_library** command for checking the physical library.

SYNTAX

```
status set_check_library_physical_options
   [-tech_consistency]
   [-view_cmp]
   [-same_name_cell]
   [-signal_em]
   [-antenna]
   [-rectilinear_cell]
   [-physical_only_cell]
   [-phys_property {place route}]
   [-routeability]
   [-tech]
   [-drc]
   [-metal_density]
   [-cells cellname_list]
   [-reset]
   [-all]
```

Data Types

```
   cellname_list  list
```

ARGUMENTS

-tech_consistency

Checks for technology data consistency between the main or design library specified by command check_library and each associated reference library. It checks the libraries for missing layer data and mismatched technology data.

-view_cmp

Checks for the existence of CEL and FRAM views in the library and for mismatched timestamps between CEL view and FRAM view.

In the report table for missing views, the CEL and FRAM columns list the cell name, version number, and an X marks the view that is missing. In the report table for mismatched views, the CEL version and FRAM version columns list the version numbers, and CEL and FRAM modified time lists the time when

the view is last modified. If a cell has an earlier FRAM view than its CEL view, it is marked as mismatched. The time checked is the internal view creation or modification time in the Milkyway database, not the Unix time. No mismatch check is performed on the cell content. If you read FRAM views from LEF and then stream in CEL views, the views are shown as mismatched. You can ignore this report in this case.

-same_name_cell

Checks for cells with identical names among the specified main or design library and all reference libraries linked to the specified library. If there are cells with the same name in multiple reference libraries, the tool uses the first cell in the reference control file and ignores the remaining cells.

-signal_em

Checks for the signal electromigration rule (current model and model type) for each routing layer.

-antenna

Checks for missing antenna properties for cells and antenna rules in the layers.

In the missing antenna property table, it lists cell names and pin names missing the antenna property and property type that is missing. If an input pin is missing the gate size, it is counted as missing the property. If an output pin is missing diode protection, it is counted as missing the property. If a macro is missing the hierarchical antenna property, then it is counted as missing the property. It is considered best practice to specify hierarchical antenna properties for macros. This option reports mode, diode mode, default metal ratio, default cut ratio and maximum ratio for antenna rule.

-rectilinear_cell

Checks and reports the cell names, types and coordinates of cells with rectilinear boundaries.

-physical_only_cell

Checks and reports the cell names, types, and properties of physical-only cells.

Physical-only cells are filler cells with and without metal, corner pad cells, image cells, flip chip pad cells, chip cells, I/O pad cells, and cover cells.

-phys_property {place route}

Checks and reports the physical properties for the standard cells.

When you specify **-phys_property place**, the tool checks and reports the placement properties for each standard cell. The placement properties include

- The place and route boundary represented by its lower-left and upper-right coordinates.
- Cell height relative to the unit tile height, such as 1xH for a single-height cell.
- Coordinate, the bottom-left location for a single-height cell or the left locations at each unit height for a multi-height cell.
- Tile pattern that represents all possible cell orientations and can have combinations of R0, R90, R160, R0_MX, R0_MY, 90_MX, and R90_MY.
- Remarks that note that the place and route boundary mismatches the tile pattern. If mismatched, use the **cmSetMultiHeightProperty** command to set tile patterns for multi-height cells.

For macros, it reports the cell boundary and height. When **place** is specified, the tool also lists the main library and its reference library names with paths, if any, and the unit tiles names and sizes. The unit tile size is reported in the format of width x height. If no tile size is reported, the library has no unit tile and the unit tile in the main library is used in the design.

When you specify **-phys_property route**, the tool checks and reports on the routing properties for each routing layer, including the preferred direction, track direction, offset, pitch, and remarks. The remarks column shows OK if the offset is 0 or half pitch; otherwise, it marks pitch=0 or offset!=0.5*pitch.

-routeability

Checks the physical pin on-track accessibility and the quality of the defined wire tracks.

It reports the total number of pins without optimal on-track routability and lists the pin names, directions, layers, and tracks for each cell with this issue.

In the track column, an H denotes that the pin is not accessible on a horizontal track, a V denotes the pin is not accessible on a vertical track, and H&V denotes that the pin is not accessible on both horizontal and vertical tracks. If a pin is reported as not accessible on a track, the pin might be routed by an off-track wire during detail routing. If too many pins do not have on-track routability, adjust the offset values of the wire track (0 or half pitch preferred) and rerun the **axgDefineWireTrack** or **create_lib_track** command to reduce the number of pins without optimal accessibility.

If routability checking is not successful, the cause might be no unit tile or incorrect technology data such as illegal fatWire threshold for some layers, illegal vias or cut layers, illegal layer number, and undefined rules.

It is possible that a track intersection is inside a via region but there is still not optimal routability because routability checking considers the worst case, that is, a via is already dropped in a neighboring pin's via region, which makes the via spot that is from the via region of the pin being checked smaller.

-tech

Checks for the technology data in the specified library.

This check is different from the **-tech_consistency** check because it checks the technology data in a single library. The messages issued during the check are similar to those issued during library creation or technology data replacement.

This option requires that the directory in which the library resides has write permissions. If the library is not writable, the tool copies the library to local first, and then checks the local library.

-drc

Specifies DRC checks for cells (FRAM view) in the specified library. Basically, it checks design rules such as wire minWidth, minEdgeLength, minEnclosedArea, minSpacing, cutWidth, cutSpacing, and minEnclosure. It lists the cells with DRC violations in a table with columns showing the cell name, type and error cell. Check the error cells for details of DRC violations. This option requires that the directory where the library resides to have write permission..If the library is not writable, it is copied to local first, and then the local library is checked.

-metal_density

Checks macro metal density data for the cells specified in option -cells.

It checks if a cell has at least one of the following errors:

- Data in the CEL view and FRAM view is inconsistent
- Metal density windows do not cover the entire area on a layer. It lists all the cells with metal density errors in a table with columns showing the cell name, data inconsistency between CEL view and FRAM view, and on which layer the density windows don't cover entire area.

-cells *cellname_list*

Specifies a list of cell names. If not specified, all cells in the library are checked.

You can use this option only with the **-routeability**, **-antenna**, **-rectilinear_cell**, **-phys_property place**, **-metal_density**, and **-view_cmp** options. If you use this option with other options, it is ignored.

-all

Performs all of the checks for the library.

-reset

Resets the options to default, which is that no checking is performed. All other options are ignored if used with -reset.

DESCRIPTION

The **set_check_library_physical_options** command sets options for checking the physical library by **check_library** command. Run this command before running the **check_library** command. If you do not specify any options with the **set_check_library_physical_options** command, the **check_library** command does not perform any checking.

The command returns a status that indicates success or failure.

EXAMPLES

The following example performs the technology consistency checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -tech_consistency
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Jan 23 11:34:34 2007
```

```
#BEGIN_CHECK_TECH_CONSISTENCY

   Errors found in technology consistency checking. (MWLIBP-328)
   Detailed technology consistency messages saved in Milkyway log file.
   Technology consistency checking is done.

#END_CHECK_TECH_CONSISTENCY

#END_CHECK_LIBRARY
```

The following example performs the view comparison checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -view_cmp
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Date and time:       Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_VIEWCMP

   Total number of cells missing CEL view:  0 (out of 997)
   Total number of cells missing FRAM view: 1 (out of 997)
   Total number of cells with mismatched view (CEL vs. FRAM):   1 (out of 997)

   X - cell missing view in the Table

        List of cells with missing views
  ------------------------------------------------------
  Cell Name             CEL                  FRAM
  ------------------------------------------------------
  cell_1                cell_1:1             X
  ------------------------------------------------------

               List of cells with mismatched views
  ------------------------------------------------------------------------------
                             CEL          CEL           FRAM         FRAM
  Cell name                Version   Modified time     Version   Modified time
  ------------------------------------------------------------------------------
  XOR11111111111111           1    Aug 19 11:46:55 2006   1    Aug 18 11:46:54 2006
  ------------------------------------------------------------------------------

#END_CHECK_VIEWCMP

#END_CHECK_LIBRARY
```

The following example performs the same name cell checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -same_name_cell
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
```

```
   Date and time:       Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_SAMENAMECELL

   Total number of cells with same names:   2 (out of 193)

        List of cells with same names
  ------------------------------------------------------------
   Cell Name                Library list
  ------------------------------------------------------------
   XOR3                     mainlib ref ref1 ref3 ref5
   DFF                      ref ref1 ref3
  ------------------------------------------------------------

#END_CHECK_SAMENAMECELL

#END_CHECK_LIBRARY
```

The following example performs the signal electromigration checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -signal_em
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Jan 23 11:34:34 2007

#BEGIN_CHECK_SIGNALEM

               List of signal EM data
   ------------------------------------------------------------
     Layer name          Current model          Model type
   ------------------------------------------------------------
     METAL1              peak                   table
     METAL1              rms                    table
     METAL1              static                 table
   ------------------------------------------------------------

#END_CHECK_SIGNALEM

#END_CHECK_LIBRARY
```

The following example performs the antenna checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -antenna
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_ANTENNA
```

```
                        List of antenna rules
 ---------------------------------------------------------------------
                                     Default     Default
  Layer Name    Mode   Diode mode   metal ratio  cut ratio   Max ratio
 ---------------------------------------------------------------------
  M1             1         3           500.00      20.00      500.00
  M2             1         3           500.00      20.00      500.00
 ---------------------------------------------------------------------

    Total number of cells missing hierarchical antenna properties:       1 (out of 1)

#END_CHECK_ANTENNA

#END_CHECK_LIBRARY
```

The following example performs the rectilinear cell checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -rectilinear_cell
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_RECTILINEARCELL

   Total number of rectilinear cells:  1 (out of 53)

    List of cells with rectilinear boundaries
-------------------------------------------------------------------------
Cell Name   Cell type  Number of points                  Coordinate
-------------------------------------------------------------------------
datapath    Macro            17    ( 78.750,   0.000) ( 78.750, 490.760)
                                   ( 44.435, 490.760) ( 44.435, 915.035)
                                   ( 27.440, 915.035) ( 27.440,1143.605)
                                   (  0.000,1143.605) (  0.000,1313.200)
                                   (677.295,1313.200) (677.295,1143.605)
                                   (649.855,1143.605) (649.855, 915.035)
                                   (632.860, 915.035) (632.860, 490.760)
                                   (598.545, 490.760) (598.545,   0.000)
                                   (  0.017,   0.000)
-------------------------------------------------------------------------

#END_CHECK_RECTILINEARCELL

#END_CHECK_LIBRARY
```

The following example performs the physical-only cell checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -physical_only_cell
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY
```

```
   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_PHYSICALONLYCELL

   Total number of physical only cells: 1 (out of 53)

              List of physical only cells
  ------------------------------------------------------------
   Cell Name            Cell type              Property
  ------------------------------------------------------------
   FILL8                FillerCell             With metal
  ------------------------------------------------------------

#END_CHECK_PHYSICALONLYCELL

#END_CHECK_LIBRARY
```

The following example performs the physical property checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -phys_property {place route}
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Jan 23 11:34:34 2007

#BEGIN_CHECK_PHYSICALPROPERTY

                List of main and reference libraries
-------------------------------------------------------------------------
Library name       Path             Unit tile  Tile size
-------------------------------------------------------------------------
test.mw            /usr/library     unit       1.800x14.400
reflib1            /usr/library     unit       1.800x14.400
-------------------------------------------------------------------------

               List of tile patterns
-------------------------------------------------------------------------
Tile pattern   Unit tile   Location   Orientation   Tile size
-------------------------------------------------------------------------
1              unit        (0,0)      R0|R0_MX      1.800x14.400
-------------------------------------------------------------------------

                List of placement properties
-------------------------------------------------------------------------
Cell name   PR boundary      Height  Symmetry  Orientation   Tile pattern
Remarks
-------------------------------------------------------------------------
DFFX1       (0,0)(11.2,5.04)  1xH    x         R0|R0_MY|R180 1
-------------------------------------------------------------------------

                List of routing properties
  ------------------------------------------------------------
             Preferred      Track
   Layer     direction    direction   Offset    Pitch   Remarks
  ------------------------------------------------------------
   METAL1         H            H       0.280     0.560   OK
   METAL2         V            V       0.330     0.660   OK
```

```
  ----------------------------------------------------------------

#END_CHECK_PHYSICALPROPERTY

#END_CHECK_LIBRARY
```

The following example performs the routability checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -routeability
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_ROUTEABILITY

   Total number of pins without on-track routeability: 1 (out of 53)

              List of pins without optimal routeability
  ------------------------------------------------------------
   Cell Name          Pin Name  Layer     Direction   Track
  ------------------------------------------------------------
   ADDF_1             CIN       METAL2    Input       H&V
  ------------------------------------------------------------

#END_CHECK_ROUTEABILITY

#END_CHECK_LIBRARY
```

The following example performs the technology checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -tech
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_TECH

Warning: Layer 'METAL1' has a pitch 0.41 that does not match the recommended
 wire-to-via pitch 0.405. (TFCHK-049)
Warning: ContactCode 'CONT1' is missing the attribute 'unitMinResistance'.
 (line 5559) (TFCHK-014)

#END_CHECK_TECH

#END_CHECK_LIBRARY
```

The following example performs the DRC checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -drc
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_DRC

   Total number of cells with DRC violations:   1 (out of 1241)

               List of cells with DRC violation
  ------------------------------------------------------------
   Cell name                   Cell type      Error cell
  ------------------------------------------------------------
   ONEPIN                      Standard       ONEPIN.err
  ------------------------------------------------------------

#END_CHECK_DRC

#END_CHECK_LIBRARY
```

The following example performs the metal density checks on the test.mw library and shows example output for the check.

```
prompt> set_check_library_physical_options -metal_density
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Sun May 27 19:26:55 2007

#BEGIN_CHECK_MACROMETALDENSITY

   Total number of cells with metal density error:   2 (out of 2)

               List of cells with metal density error
  ------------------------------------------------------------------
   Cell name          Inconsistency       Not cover entire area
  ------------------------------------------------------------------
   ADDFX1MTR          Y                   M4
   OA21X1MTR          N                   M1
                                          M2
  ------------------------------------------------------------------

#END_CHECK_MACROMETALDENSITY

#END_CHECK_LIBRARY
```

The following example checks all data for the test.mw library.

```
prompt> set_check_library_physical_options -all
prompt> check_library -mw_lib_name test.mw
1
```

The following example shows the **check_library** output:

```
#BEGIN_CHECK_LIBRARY

   Main library name:   /usr/library/test.mw
   Check date and time: Tue Nov 28 17:59:07 2006

#BEGIN_CHECK_TECH_CONSISTENCY

   No technology inconsistency found.

#END_CHECK_TECH_CONSISTENCY

#BEGIN_CHECK_TECH

Warning: Layer 'METAL1' has a pitch 0.41 that does not match the recommended
 wire-to-via pitch 0.405. (TFCHK-049)
Warning: ContactCode 'CONT1' is missing the attribute 'unitMinResistance'.
 (line 5559) (TFCHK-014)

#END_CHECK_TECH

#BEGIN_CHECK_VIEWCMP

   Total number of cells missing CEL view:  0 (out of 997)
   Total number of cells missing FRAM view: 1 (out of 997)
   Total number of cells with mismatched view (CEL vs. FRAM):   0 (out of 997)

   X - cell missing view in the Table

        List of cells with missing views
  ------------------------------------------------------
  Cell Name          CEL                 FRAM
  ------------------------------------------------------
  cell_1             cell_1:1            X
  ------------------------------------------------------

#END_CHECK_VIEWCMP

#BEGIN_CHECK_SAMENAMECELL

   Total number of cells with same names:   2 (out of 193)

        List of cells with same names
  ------------------------------------------------------------
   Cell Name                Library list
  ------------------------------------------------------------
   XOR3                     mainlib ref ref1 ref3 ref5
   DFF                      ref ref1 ref3
  ------------------------------------------------------------

#END_CHECK_SAMENAMECELL

#BEGIN_CHECK_SIGNALEM

               List of signal EM data
   ------------------------------------------------------------------
     Layer name          Current model          Model type
   ------------------------------------------------------------------
     METAL1              peak                   table
     METAL1              rms                    table
     METAL1              static                 table
   ------------------------------------------------------------------

#END_CHECK_SIGNALEM
```

```
#BEGIN_CHECK_ANTENNA

                        List of antenna rules
 ------------------------------------------------------------------------
                                     Default     Default
  Layer Name    Mode   Diode mode   metal ratio  cut ratio   Max ratio
 ------------------------------------------------------------------------
  M1             1         3           500.00      20.00      500.00
  M2             1         3           500.00      20.00      500.00
 ------------------------------------------------------------------------

   Total number of cells missing hierarchical antenna properties:       1 (out of 53)

               List of cells missing antenna property
  ---------------------------------------------------------------------------
   Cell name             Cell type  Missing property     Pin name(s)
  ---------------------------------------------------------------------------
   ram256x27             Macro      HierAntenna          Q[16] Q[17] Q[18] Q[19]
  ---------------------------------------------------------------------------

#END_CHECK_ANTENNA

#BEGIN_CHECK_RECTILINEARCELL

   Total number of rectilinear cells:  1 (out of 53)

    List of cells with rectilinear boundaries
-------------------------------------------------------------------------
Cell Name   Cell type  Number of points                   Coordinate
-------------------------------------------------------------------------
datapath    Macro            17    ( 78.750,   0.000) ( 78.750, 490.760)
                                   ( 44.435, 490.760) ( 44.435, 915.035)
                                   ( 27.440, 915.035) ( 27.440,1143.605)
                                   (  0.000,1143.605) (  0.000,1313.200)
                                   (677.295,1313.200) (677.295,1143.605)
                                   (649.855,1143.605) (649.855, 915.035)
                                   (632.860, 915.035) (632.860, 490.760)
                                   (598.545, 490.760) (598.545,   0.000)
                                   (  0.017,   0.000)
-------------------------------------------------------------------------

#END_CHECK_RECTILINEARCELL

#BEGIN_CHECK_PHYSICALONLYCELL

   Total number of physical only cells: 1 (out of 53)

               List of physical only cells
  ------------------------------------------------------------
   Cell Name            Cell type              Property
  ------------------------------------------------------------
   FILL8                FillerCell             With metal
  ------------------------------------------------------------

#END_CHECK_PHYSICALONLYCELL

#BEGIN_CHECK_PHYSICALPROPERTY

                List of main and reference libraries
-------------------------------------------------------------------------------
Library name       Path                                   Unit tile  Tile size
-------------------------------------------------------------------------------
test.mw            /usr/library                           unit       1.800x14.400
```

```
reflib1            /usr/library                              unit       1.800x14.400
-------------------------------------------------------------------------------

               List of tile patterns
-------------------------------------------------------------------------------
Tile pattern   Unit tile   Location   Orientation   Tile size
-------------------------------------------------------------------------------
1              unit        (0,0)      R0|R0_MX      1.800x14.400
-------------------------------------------------------------------------------

                List of placement properties
-------------------------------------------------------------------------------
Cell name   PR boundary      Height  Symmetry  Orientation   Tile pattern
Remarks
-------------------------------------------------------------------------------
DFFX1       (0,0)(11.2,5.04)  1xH    x         R0|R0_MY|R180      1
-------------------------------------------------------------------------------

                List of routing properties
  ---------------------------------------------------------------------
             Preferred      Track
   Layer     direction    direction   Offset    Pitch   Remarks
  ---------------------------------------------------------------------
   METAL1         H            H       0.280     0.560   OK
   METAL2         V            V       0.330     0.660   OK
  ---------------------------------------------------------------------

#END_CHECK_PHYSICALPROPERTY

#BEGIN_CHECK_ROUTEABILITY

   Total number of pins without on-track routeability: 1 (out of 53)

               List of pins without optimal routeability
  ---------------------------------------------------------------
   Cell Name          Pin Name  Layer     Direction   Track
  ---------------------------------------------------------------
   ADDF_1             CIN       METAL2    Input       H&V
  ---------------------------------------------------------------

#END_CHECK_ROUTEABILITY

#BEGIN_CHECK_DRC

   Total number of cells with DRC violations:   1 (out of 1241)

               List of cells with DRC violation
  ---------------------------------------------------------------
   Cell name                    Cell type      Error cell
  ---------------------------------------------------------------
   ONEPIN                       Standard       ONEPIN.err
  ---------------------------------------------------------------

#END_CHECK_DRC

#END_CHECK_LIBRARY
1
```

SEE ALSO

check_library(2)
report_check_library_options(2)

set_hierarchy_separator

Sets the hierarchy separator character to the specified value.

SYNTAX

```
status set_hierarchy_separator char
```

char

ARGUMENTS

char

Specifies the character to use as the hierarchy separator.

DESCRIPTION

The **set_hierarchy_separator** command defines the character that delimits hierarchy.

Please see the man page for the **hierarchy_separator** variable. The following two commands are the same:

```
prompt> set_hierarchy_separator "/"
prompt> set hierarchy_separator "/"
```

EXAMPLES

The following example sets the hierarchy separator character to @.

```
prompt> set_hierarchy_separator "@"
```

SEE ALSO

hierarchy_separator(3)

set_mw_lib_reference

Sets the reference library for the Milkyway library.

SYNTAX

```
status_value set_mw_lib_reference
   [-mw_reference_library lib_list]
   [-reference_control_file file_name]
   libName
```

Data Types

```
   lib_list   string
   file_name  string
   libName    string
```

ARGUMENTS

-mw_reference_library *lib_list*

Specifies a list of libraries to be set as reference libraries for the Milkyway library.

This argument and **-reference_control_file** are mutually exclusive.

-reference_control_file *file_name*

Specifies a reference control file containing information to set the reference libraries for the Milkyway library.

This argument and **-mw_reference_library** are mutually exclusive.

libName

Specifies the Milkyway library to be worked on.

DESCRIPTION

Sets or changes the reference libraries for the Milkyway library. The **-reference_control_file** and **-mw_reference_library** options are mutually exclusive, and at least one of the two must be specified.

A status indicating success or failure is returned.

EXAMPLES

The following example sets the reference libraries with a list:

```
prompt> set_mw_lib_reference -mw_reference_library {./lib/ref1 ./lib/ref2} design
```

SEE ALSO

```
create_mw_lib(2)
set_mw_technology_file(2)
```

set_mw_technology_file

Specifies the technology file of the Milkyway library.

SYNTAX

```
status_value set_mw_technology_file
   [-technology tech_file]
   [-plib plib_file]
   [-alf alf_file]
   libName
```

Data Types

```
   tech_file  string
   plib_file  string
   alf_file  string
   libName    string
```

ARGUMENTS

-technology *tech_file*

Specifies a new technology file to use with the Milkyway library. The old technology information is completely replaced. Ensure that the new technology information is compatible, If the new technology information is not compatible, you will have to recreate the Milkyway library. The **-technology** and **-plib** and **-alf** options are mutually exclusive.

-plib *plib_file*

Specifies a new .plib file to use with the Milkyway library. If the .plib file is an incremental .plib file, the command loads the technology information in the .plib file incrementally and updates the open library. If it is a complete technology .plib file, the library technology information in the open library is replaced. Ensure that the new technology information is compatible. If the new technology information is not compatible, you will have to recreate the Milkyway library. The **-technology** and **-plib** and **-alf** options are mutually exclusive.

-alf *alf_file*

Specifies a new advanced library format (ALF) file to use with the Milkyway library. Use this option to import signal EM data into the library. Existing signal EM information in the library is completely replaced by the data in the new ALF file. Ensure that the new signal EM information is compatible with library

technology information. Signal EM data in the ALF file is updated to the library immediately and it does not have any impact on the open design. The **-technology** and **-plib** and **-alf** options are mutually exclusive.

libName

Specifies the Milkyway library to be updated. The value of *mw_lib* must be a valid library name.

DESCRIPTION

This command sets the technology file for a Milkyway library.

The command returns a status indicating success or failure.

EXAMPLES

The following example sets the technology file of the library design to *new.tf*:

```
prompt> set_mw_technology_file -technology new.tf design
1
```

SEE ALSO

```
close_mw_lib(2)
copy_mw_lib(2)
create_mw_lib(2)
current_mw_lib(2)
open_mw_lib(2)
rename_mw_lib(2)
set_mw_lib_reference(2)
write_mw_lib_files(2)
```

set_preferred_routing_direction

Sets the preferred routing direction for a layer.

SYNTAX

```
int set_preferred_routing_direction
   -layer layer_name
   -dir direction

layer_name string
direction string
```

ARGUMENTS

-layer *layer_name*

Specifies the routing layer name for which perferred routing direction is to be set.

-dir *direction*

Specifies the perferred routing direction. The valid values are **horizontal** and **vertical**; specify either.

DESCRIPTION

This command sets perferred routing direction for the routing layer.

EXAMPLES

The following example sets preferred routing direction for a routing layer named *Metal1*.

```
prompt> set_preferred_routing_direction -layer Metal1 -dir horizontal
```

SEE ALSO

unset_preferred_routing_direction(2)

set_user_grid

Set user grid for this session

SYNTAX

```
status
set_user_grid
   [-x_offset x_offset]
   [-y_offset y_offset]
   [-x_step x_step]
   [-y_step y_step]
   [-user_grid grid]
   [-x_get_from_layer x_layer]
   [-y_get_from_layer y_layer]
   [-reset]
   [-design]
```

ARGUMENTS

-x_offset *x_offset*

Specifies the horizontal (X) distance (in user units) by which you want to offset the cursor snap-to points from the grid point. The value must be multiple of litho grid. This option is exclusive with -user_grid and -x_get_from_layer option. You can only use one of them.

-y_offset *y_offset*

Specifies the vertical (Y) distance (in user units) by which you want to offset the cursor snap-to points from the grid point. The value must be multiple of litho grid. This option is exclusive with -user_grid and -y_get_from_layer option. You can only use one of them.

-x_step *x_step*

Specifies the horizontal (X) distance (in user units) you want between cursor snap-to points. The value must be multiple of litho grid. This option is exclusive with -user_grid and -x_get_from_layer option. You can only use one of them.

-y_step *y_step*

Specifies the vertical (Y) distance (in user units) you want between cursor snap-to point. The value must be multiple of litho grid. This option is exclusive with -user_grid and y_get_from_layer option. You can only use one of them.

-user_grid *{{x_offset y_offset} {x_step y_step}}*

Specifies the user grid in list representation. The element value must be multiple of litho grid. This option is exclusive with all other options.

-x_get_from_layer *x_layer*

Indicate to have the tool automatically calculate the user grid x_offset x_step from the wire track on the specified layer. The value could be layer name or layer number or layer collection. This option is exclusive with -x_offset -x_step and -user_grid option. You can only use one of them.

-y_get_from_layer *y_layer*

Indicate to have the tool automatically calculate the user grid y_offset y_step from the wire track on the specified layer. The value could be layer name or layer number or layer collection. This option is exclusive with -y_offset -y_step and -user_grid option. You can only use one of them.

-reset

Reset the user grid to litho grid. This option can only be used with design option.

design

Specifies in which design to set the user grid information. If this option is not specified, it will set global user grid information which is used for default value of successive run of this command.

DESCRIPTION

This command set user grid for this session of the tool. You can use the -user_grid option to set the user grid inone shot. Or else, you can use -x_offset, -y_offset, -x_step, -y_step to specify individual part of the user grid. If any of these four options are not specified, then the corresponding value is not changed. Finally, you can use -x_get_from_layer and/or -y_get_from_layer to have the tool automatically calculate the user_grid from the wire track on the given layer.

The user grid must be multiple of litho grid. If it's not, this command will fail with error message MWUI-212.

If design is not specified, then this command will set global user grid. The global user grid will be used as default value for design specific user grid until you explicitly set user grid on the design.

EXAMPLES

The following example set global user grid in one shot:

```
prompt> set_user_grid -user_grid {{0 0} {0.010 0.010}}
1
```

The following example set x_offset and y_offset of the global user grid only:

```
prompt> set_user_grid -x_offset 0.5 -y_offset 0.6
1
```

The following example set user grid from wire track information on given design top.

```
prompt> set_user_grid -x_get_from_layer M1 -y_get_from_layer M2 top
1
```

The following example failed because the value specified is not multiple of litho grid, suppose litho grid is 0.001:

```
prompt> set_user_grid -x_offset 0.0005
Error: Value '0.0005' is not multiple of litho grid '0.001'. (MWUI-212)
0
```

SEE ALSO

get_user_grid(2)

set_via_array_size

Modifies the array size of an existing via or via array.

SYNTAX

```
collection set_via_array_size
   -array_size {row col}
   via_collection
```

Data Types

```
   via_collection          collection
```

ARGUMENTS

-array_size {*row col*}

Specifies the new array_size of the via/via_array. The value of *row* and *col* must be positive integer. If you specify {1 1} to a via_array, the via_array will be converted to a via. If you specify another value except {1 1} to a via, the via will be converted to a via_array.

You must specify this option.

via_collection

Specifies a list of via/via_arrays that to be assigned with new array_size value.

DESCRIPTION

The **set_via_array_size** command enables you to convert a via to a via_array and vice versa. You may also use this command to change the value of array_size of a via_array. The return value is a collection contains the modified via/via_array.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example modifies a via to a via_array with array_size {2 3}

icc_shell> **set_via_array_size -array_size {2 3} VIA#4075 {VIA_ARRAY#86054}**

SEE ALSO

```
create_via(2)
get_vias(2)
remove_via(2)
```

sizeof_collection

Returns the number of objects in a collection.

SYNTAX

```
int sizeof_collection collection1

collection1  collection
```

ARGUMENTS

collection1

Specifies the collection for which to get the number of objects. If the empty collection (empty string) is used for the *collection1* argument, the command returns 0.

DESCRIPTION

The **sizeof_collection** command is an efficient mechanism for determining the number of objects in a collection.

EXAMPLES

The following example shows a simple way to find out how many objects matched a particular pattern and filter in a **get_cells** command.

```
prompt> echo "Number of hierarchical cells: \
?                [sizeof_collection [get_cells * -hier \
?                   -filter "is_hierarchical == true"]]"
Number of hierarchical cells is: 10
```

The following example shows what happens when the argument to **sizeof_collection** results in an empty collection.

```
prompt> set s1 [get_cells *]
{"u1", "u2", "u3"}

prompt> set ssize [filter_collection $s1 "area < 0"]

prompt> echo "Cells with area < 0: [sizeof_collection $ssize]"
Cells with area < 0: 0

prompt> unset s1
```

SEE ALSO

collections(2)
filter_collection(2)

sort_collection

Sorts a collection based on one or more attributes, resulting in a new, sorted collection. The sort is ascending by default.

SYNTAX

```
collection sort_collection
   [-descending]
   collection1
   criteria


collection1   collection
criteria      list
```

ARGUMENTS

-descending

Indicates that the collection is to be sorted in reverse order. By default, the sort proceeds in ascending order.

collection1

Specifies the collection to be sorted.

criteria

Specifies a list of one or more application or user-defined attributes to use as sort keys.

DESCRIPTION

You can use the **sort_collection** command to order the objects in a collection based on one or more attributes. For example, to get a collection of leaf cells increasing alphabetically, followed by hierarchical cells increasing alphabetically, sort the collection of cells using the **is_hierarchical** and **full_name** attributes as *criteria*.

In an ascending sort, Boolean attributes are sorted with those objects first that have the attribute set to *false*, followed by the objects that have the attribute set to *true*. In the case of a sparse attribute, objects that have the attribute come first, followed by the objects that do not have the attribute.

Sorts are ascending by default. The **-descending** option reverses the order of the objects.

EXAMPLES

The following example sorts a collection of cells based on hierarchy, and adds a second key to list them alphabetically. In this example, cells i1 and i2 are hierarchical, and u1 and u2 are leaf cells. Because **is_hierarchical** is a Boolean attribute, those objects with the attribute set to *false* are listed first in the sorted collection.

```
prompt> set zcells [get_cells {o2 i2 o1 i1}]
{"o1", "i2", "o1", "i1"}

prompt> set zsort [sort_collection $zc {is_hierarchical full_name}]
{"o1", "o2", "i1", "i2"}
```

SEE ALSO

collections(2)

suppress_message

Disables printing of one or more informational or warning messages.

SYNTAX

```
string suppress_message [message_list]

message_list list
```

ARGUMENTS

message_list

Specifies a list of messages to suppress.

DESCRIPTION

This command provides a mechanism to disable the printing of messages. You can suppress only informational and warning messages. The result of using this command is always an empty string.

You can suppress a given message more than once. To enable a message, you must use the **unsuppress_message** command to unsuppress it as many times as you suppressed it. The **print_suppressed_messages** command displays the currently-suppressed messages.

EXAMPLES

The following example shows how to suppress the **CMD-029** message, which displays when the argument to the **unalias** command does not match any existing aliases.

```
prompt> unalias q*
```

```
Warning: no aliases matched 'q*' (CMD-029)
prompt> suppress_message CMD-029
prompt> unalias q*
prompt>
```

SEE ALSO

print_suppressed_messages(2)
unsuppress_message(2)

undefine_user_attribute

Undefines a user-defined attribute.

SYNTAX

```
int undefine_user_attribute
   -class class_list
   [-quiet]
   attr_name
```

Data Types

class_list list *attr_name* string

ARGUMENTS

-class *class_list*

Specifies the list of class names for the user-defined *attr_name* attribute. The following is a list of valid values for the elements of *class_list*:

```
design
port
cell
pin
net
lib
lib_cell
lib_pin
clock
bound
net_shape
placement_blockage
route_guide
route_shape
terminal
text
```

-quiet

Turns off the warning message that would otherwise be issued if the attribute or classes are improper.

attr_name

Specifies the name of the attribute.

DESCRIPTION

This command undefines a user-defined attribute. The return value is set to **1** if the operation is successful; otherwise, it is set to **0**.

Note that a user-defined attribute can be undefined only if there is no object with the specified attribute in the database.

EXAMPLES

The following example undefines an attribute named *attr_s* from a net named *net*.

```
prompt> remove_attribute -class net * attr_s -quiet
prompt> undefine_user_attribute -class net attr_s
Info:Removing user-defined attribute 'attr_s' on class 'net'.
1
```

SEE ALSO

```
define_user_attribute(2)
get_attribute(2)
list_attributes(2)
remove_attribute(2)
set_attribute(2)
```

unset_preferred_routing_direction

Unsets the preferred routing direction for a layer.

SYNTAX

```
int unset_preferred_routing_direction
   -layer layer_name

layer_name string
```

ARGUMENTS

-layer *layer_name*

Specifies the routing layer name for which perferred routing direction is to be unset.

DESCRIPTION

This command unsets perferred routing direction for the routing layer to be undefined.

EXAMPLES

The following example unsets preferred routing direction for a routing layer named *Metal1*.

```
prompt> unset_preferred_routing_direction -layer Metal1
```

SEE ALSO

set_preferred_routing_direction(2)

unsuppress_message

Enables printing of one or more suppressed informational or warning messages.

SYNTAX

```
string unsuppress_message [messages]

messages list
```

ARGUMENTS

messages

Specifies a list of messages to enable.

DESCRIPTION

This command provides a mechanism to re-enable the printing of messages that have been suppressed by using the **suppress_message** command. You can suppress only informational and warning messages, so the **unsuppress_message** command is only useful for informational and warning messages. The result of using this command is always an empty string.

You can suppress a given message more than once. To enable a message, you must unsuppress a message as many times as you used the **suppress_message** command to suppress it. The **print_suppressed_messages** command displays currently-suppressed messages.

EXAMPLES

The following example shows how to re-enable a suppressed **CMD-029** message, which would, if unsuppressed, display when the argument to the **unalias** command does not match any existing aliases.

This example assumes that there are no aliases beginning with *q*.

```
prompt> unalias q*
prompt> unsuppress_message CMD-029
prompt> unalias q*
Warning: no aliases matched 'q*' (CMD-029)
prompt>
```

SEE ALSO

print_suppressed_messages(2)
suppress_message(2)

update_mw_bound

Modifies information for an existing bound.

SYNTAX

```
status_value update_mw_bound
   -add cells | -remove cells
   bound
```

ARGUMENTS

-add *cells*

Specifies the cells to be added to the bound.

-remove *cells*

Specifies the cells to be removed from the bound.

***bound*"**

Specifies a bound to be updated.

DESCRIPTION

This command modifies the bound information. To add one or more cells to the specified bound, include the **-add** option and specify the bound. You must specify a bound to remove cells from, and only the cells that belong to that bound will be removed. The **-add** and **-remove** options are mutually exclusive. This command returns a 1 if successful.

EXAMPLES

The following example adds cells to the specified bound.

```
prompt> update_mw_bound MB0 -add [get_cells r*]
1
```

The following example removes cells from the specified bound.

```
prompt> update_mw_bound MB0 -remove [get_cells r*]
1
```

SEE ALSO

create_mw_bound(2)
get_bounds(2)
remove_mw_bounds(2)

update_mw_port_by_db

Updates pin attributes of all cells in one Milkyway library by using information from .db files.

SYNTAX

```
status_value update_mw_port_by_db
   -mw_lib lib_name
   -db_file {db_file list}
   [-skip_direction]
   [-bias_pg | -create_bias_text]
   [-update_bus]
   [-update_analog_pin]
   [-update_antenna_diode_type]
```

Data Types

```
   lib_name string
   db_file list list of string
```

ARGUMENTS

-mw_lib *lib_name*

Specifies the Milkyway library to be updated. The value of *lib_name* can be a library name or a one-element collection of a library. This option must be specified.

-db_file *db_file list*

Specifies the names of the .db files to be used to annotate the Milkyway library. The value of *db_file list* can be a .db file name or a collection of .db file names. This option must be specified.

-skip_direction

Causes the command to skip updating the signal pin direction information from the .db files to the Milkyway library. By default, the command updates the signal pin direction information.

-bias_pg

Causes the command to update the cell bias PG pin information from the .db files to the Milkyway library. Using this option updates the corresponding pin type to bias PG type if it exists in the FRAM view; otherwise, it creates a corresponding dummy port in FRAM with the bias PG type, without creating

any real geometry.

```
Bias PG type in Liberty vs. FRAM
-------------------------------------------------------------------------
        Liberty                     |        FRAM
-------------------------------------------------------------------
PG type    |  physical_connection   |   GPortTable/Query type
-------------------------------------------------------------------
nwell      |  device_layer          |   Power BiasPG
deepnwell  |  device_layer          |
-------------------------------------------------------------------
pwell      |  device_layer          |   Ground BiasPG
deeppwell  |  device_layer          |
-------------------------------------------------------------------
nwell      |  routing_pin           |   Power SecondaryPG BiasPG
deepnwell  |  routing_pin           |
-------------------------------------------------------------------
pwell      |  routing_pin           |   Ground SecondaryPG BiasPG
depppwell  |  routing_pin           |
-------------------------------------------------------------------
```

The **-bias_pg** and **-create_bias_text** options are mutually exclusive.

-create_bias_text

Creates a TEXT object showing the bias pin name on the corresponding geometry in the CEL view. The TEXT object is created on the first point or lower-left corner point of the geometry. This routine can help a pin extraction application recognize the pin. You still need to specify the correct PG type in GPortTable to correctly mark the bias pin in the FRAM view.

Only *Path/Polygon/Rectangle* type geometries are supported. The **-create_bias_text** and **-bias_pg** options are mutually exclusive.

-update_bus

Updates the bus pin information from the .db files to the FRAM view. It updates the bus index property of the pins using the bit_width defined in the .db files.

-update_analog_pin

Updates the analog pin information from the .db files to the FRAM view. It marks the corresponding pin as analog pin in the FRAM view.

-update_antenna_diode_type

Updates the antenna diode cell type from the .db files to the FRAM view. The valid diode cell type could be:

"power_and_ground" "power" "ground"

DESCRIPTION

This command updates all FRAM cells in one specified Milkyway library with information from one or more specified .db files. It can update various attributes, depending on the command options used.

By default, the command updates the signal pin direction and secondary power/ground pin information in the FRAM views. It can be used in place of the **gePrepLibs** command.

The command returns a status indicating success or failure.

EXAMPLES

The following command updates the Milkyway library called my_mw_lib with signal pin direction information, power/ground pin information, and bias PG pin information taken from the a.db and b.db files.

```
prompt> update_mw_port_by_db -mw_lib my_mw_lib -db_file {a.db b.db} \
        -bias_pg
Updated signal pin information successfully.
Updated secondary PG information successfully.
```

SEE ALSO

close_mw_lib(2)
open_mw_lib(2)
get_attribute(2)

update_voltage_area

Updates an existing voltage area by adding new shapes or specifying new physical margin guard bands.

SYNTAX

```
status update_voltage_area
   -voltage_area area_name
   [-guard_band_x margin]
   [-guard_band_y margin]
   [-coordinates {llx1 lly1 urx1 ury1}]
   [-polygons {{{x1 y1} ... {xN yN} {x1 y1}} ... }]
```

Data Types

```
   area_name  collection or list
   margin     float
   llx1       float
   lly1       float
   urx1       float
   ury1       float
   x1         float
   y1         float
   xN         float
   yN         float
```

ARGUMENTS

-voltage_area *area_name*

Specifies the voltage area affected by the command by name or collection handle.

-guard_band_x *margin*

Specifies the new horizontal (left and right) margin around the voltage area where objects are not allowed to be placed.

-guard_band_y *margin*

Specifies the new vertical (top and bottom) margin around the voltage area where objects are not allowed to be placed.

-coordinates {*llx1 lly1 urx1 ury1 ...*}*

Specifies one or more rectangles to be added to the voltage area. Each set of four values specifies the x and y coordinates of the lower-left and upper-right corners of a rectangle, in microns.

-polygons {{{*x1 y1*} ... {*xN yN*} {*x1 y1*}} ...}*

Specifies a list of polygons to be added to the voltage area. Each set of value {*x1 y1 ... xN yN x1 y1*} defines the list of points for one polygon. Each shape can be a rectangle (5 points) or a rectilinear polygon. The unit is microns.

DESCRIPTION

This command updates an existing voltage area previously created by the **create_voltage_area** command by adding new shapes or specifying new physical margin guard bands.

To change the physical guard bands around the voltage area, use the **-guard_band_x** and **-guard_band_y** options. The tool considers the guard bands as placement keepout regions.

To add regions to an existing voltage area, use the **-coordinates** or **-polygons** option.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

EXAMPLES

The following example updates the guard band for the voltage area VA1.

```
prompt> update_voltage_area -voltage_area VA1 -guard_band_x 3 -guard_band_y 5
1
```

The following commands add rectangular regions to the voltage area VA1.

```
prompt> update_voltage_area -voltage_area VA1 -coordinates { 215 215 350 350 }
1
prompt> update_voltage_area -voltage_area VA2 \
        -polygons {{100 100} {300 100} {300 200} {100 200} {100 100}}
1
```

SEE ALSO

create_voltage_area(2)
remove_voltage_area(2)
report_voltage_area(2)

```
get_voltage_areas(2)
```

write_def

Writes the design data of the specified design to a file in DEF format, including physical layout, netlist and design constrains.

SYNTAX

```
int write_def
   -output output_file_name
   [-lib library_name]
   [-topdesign design_name]
   [-version def_version]
   [-unit conversion_factor]
   [-compressed]
   [-rows_tracks_gcells]
   [-vias]
   [-all_vias]
   [-lef lef_file_name]
   [-regions_groups]
   [-components]
   [-macro ]
   [-fixed ]
   [-placed ]
   [-pins ]
   [-blockages ]
   [-specialnets ]
   [-nets ]
   [-routed_nets ]
   [-diode_pins ]
   [-scanchain ]
   [-notch_gap ]
   [-floating_metal_fill ]
   [-pg_metal_fill ]
   [-extra_dummy_track ]
   [-no_legalize ]

string  library_name
string  design_name
string  output_file_name
string  lef_file_name
```

ARGUMENTS

-output *output_file_name*

Specifies the name of the output DEF file written by **write_def**. This is a required argument.

-lib *library_name*

Specifies the name of the Milkyway library which contains the design cell. If not specified will use the current design library.

-topdesign *design_name*

Specifies the name of the design to be written to the output DEF file. If not specified will use the current design library.

-version *def_version*

Specifies the version of DEF to be written to output DEF file. Valid def version are 5.3, 5.4, 5.5 and 5.6. The default is DEF 5.5.

-unit *conversion_factor*

Specifies the value used in the DEF UNITS DISTANCE MICRONS statement. Valid values for conversion factor are 100, 200, 1000, 2000, 10000 and 20000. The values 10000 and 20000 are supported in DEF 5.6.

-compressed

Select this option to generate a gzipped format file.

-rows_tracks_gcells

Specifies that the ROW, TRACK and GCELLGRID sections are to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-vias

Specifies that the VIAS section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-all_vias

Specifies that all vias (those present in the design, as well as those present in technology file) are included in the output DEF file. If this option is not specified, only vias present within the design are written out.

-lef *lef_file_name*

Specifies that rotated vias and design specified nondefault rule are to be written to the specified LEF file. Default file name is MW.lef.inc

-regions_groups

Specifies that the REGIONS and GROUPS sections are to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-components

Specifies that the COMPONENTS section is to be included in the output DEF file. It can be combined

with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-macro

Specifies that the macro cells are to be included in the output DEF file.

-fixed

Specifies that the fixed cells are to be included in the output DEF file.

-placed

Specifies that the placed cells are to be included in the output DEF file.

If all/none of **-macro**, **-fixed**, and **-placed**, are specified, all cell instances will be output.

-pins

Specifies that the PINS section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-blockages

Specifies that the BLOCKAGES section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-specialnets

Specifies that the SPECIALNETS section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-nets

Specifies that the NETS section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-routed_nets

Specifies that only routed nets are to be included in the output DEF file.

-diode_pins

Specifies that the diode(extra) pins are to be included in the NETS section. By default, diode_pins option is on.

-scanchain

Specifies that the SCANCHAINS section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. If none of these options are specified, all DEF sections are written out.

-notch_gap

Specifies that the notch/gap is to be included in SPECIALNETS section. By default, notch_gap option is

off.

-floating_metal_fill

Specifies that the FILLS section is to be included in the output DEF file. It can be combined with options for writing out other DEF sections. By default, floating_metal_fill option is off.

-pg_metal_fill

Specifies that the P/G metal fills are to be included in SPECIALNETS section. By default, pg_metal_fill option is off.

-extra_dummy_track

Specifies that the extra dummy tracks are to be included in the output DEF file. Other EDA tools may need these dummy tracks so that other wire tracks are expanded to the boundary. By default, extra_dummy_track option is off.

-no_legalize

Specifies that name legalization should not occur. Name legalization escapes DEF special characters that do not actually have special meaning. Escaping the special character causes its special meaning to be ignored by the DEF reader. For example, if a component name contains bus bit characters, the bus bit characters will be escaped because it is not really a bus. By default, no_legalize option is off.

DESCRIPTION

The **write_def** command writes the design data of the specified design to a file in DEF format, including physical layout, netlist and scan def.

The write_def by default will not dump out all the vias of the design library. The user has to explicitly select all_vias option to dump the library vias.

The Milkyway design library and the cell should be opened prior to calling the write_def command.

EXAMPLES

The following example writes out a file *out.def* from the existing design *top*.

```
prompt> write_def  -output out.def -compressed
```

SEE ALSO

read_def(2)

write_design_settings

Reports the session variable settings previously saved with the **save_design_settings** command.

SYNTAX

```
status write_design_settings
   -library | -design
   [-output file]
```

Data Types

```
   file              string
```

ARGUMENTS

-library

Reports the design settings that were previously saved in the current library.

-design

Reports the design settings thate were previously saved in the current design cell.

-output *file*

Writes the design settings to the specified file and prevents reporting of the design settings on the screen.

DESCRIPTION

This command reports the design settings that were previously saved into the current library or design cell with the **save_design_settings** command. It does not apply those commands to the current session.

By default, the design settings are reported in the session log and console. Use the **-output** option to write the design settings to a file.

To restore and apply the saved design settings, use the **restore_design_settings** command.

EXAMPLES

The following example displays the design settings saved in the current library.

```
prompt> write_design_settings -library
set LIBRARY_LIB /usr/LIB/130nm/SMALL
set LIBRARY_DESIGN /usr/DATA/Mini/SMALL

set search_path          " . \
                                $LIBRARY_DESIGN/orig_data \
                                $LIBRARY_LIB/DB \
                                $LIBRARY_LIB/PDB \
                                $LIBRARY_LIB/RCXT  \
                          "


set physical_library   [list design1_mw.pdb REF1.pdb REF2.pdb REF3.pdb \
                        REF4.pdb REF5.pdb]

set target_library    "REF1 _Worst.db REF2_Worst.db \
                       REF3_Worst.db REF4_Worst.db \
                       REF5_Worst.db "

set link_library  "* $target_library"

set mw_reference_library " \
                                $LIBRARY_LIB/MW/REF1 \
                                $LIBRARY_LIB/MW/REF2 \
                                $LIBRARY_LIB/MW/REF3 \
                                $LIBRARY_LIB/MW/REF4 \
                                $LIBRARY_LIB/MW/REF5 \
                          "


set mw_design_library design
set design_name SMALL
set DESIGN $design_name

set mw_tech_file $LIBRARY_DESIGN/orig_data/top.tf

set mdb_tlu_plus_max_file "$LIBRARY_LIB/RCXT/worst.itf.tlu"
set mdb_tlu_plus_min_file "$LIBRARY_LIB/RCXT/best.itf.tlu"
set mdb_tlu_plus_map_file "$LIBRARY_LIB/RCXT/astro.map"

set mw_power_net VDD
set mw_ground_net VSS
set mw_logic1_net VDD
set mw_logic0_net VSS

suppress_message PSYN-024
suppress_message PSYN-039
suppress_message PSYN-025 ;# physical only cells
suppress_message PSYN-036 ;# linking logical with physical library
suppress_message PSYN-040 ;# dont touch on fixed cells
suppress_message PSYN-088 ;# dont touch on pad
suppress_message PSYN-087 ;# io pad location for port
```

```
    suppress_message OPT-1022 ;# io pad and usable for logic
    suppress_message LINT-30  ;# unconnected ports

    set bus_name_style      {[%d]}
    1
```

SEE ALSO

```
restore_design_settings(2)
save_design_settings(2)
```

write_gds

Writes the library data of the specified library to a file in GDS format.

SYNTAX

```
int write_gds
   [ -lib_name <lib_name> ]
   [ -layer_mapping <layer_mapping_file_name> ]
   [ -child_depth <child_cell_depth> ]
   [ -output_cell <cell_name> ]
   [ -skip_ref_lib_cells ]
   [ -text_factor <factor_number> ]
   [ -text_width <width_number>]
   [ -flatten_contact ]
   [ -flatten_contact_array ]
   [ -contact_prefix <contact_cell_prefix> ]
   [ -notch ]
   [ -gap ]
   [ -fill ]
   [ -force_output_outdate_fill ]
   [ -generate_instance_name_as_prop <prop_number> ]
   [ -generate_geom_property ]
   [ -strip_backslash_from_instance_net_names ]
   [ -skip_layer_mapping_for_imported_cells ]
   [ -output_pin_as_text ]
   [ -output_pin_as_geometry ]
   [ -pin_name_mag <mag_number> ]
   [ -rotate_pin_text_by_access_dir ]
   [ -output_net_as_text ]
   [ -output_net_as_property <prop_number> ]
   [ -net_name_mag <mag_number> ]
   [ -output_net_as_plex ]
   [ -long_name_length <name_length_limit>]
   [ -keep_data_type ]
   [ -output_odd_pin ]
   [ -rename_cell_name <rename_file>]
   [ -compressed ]
   <gds_file_name>

string  lib_name
string  layer_mapping_file_name
string  contact_cell_prefix
string  gds_file_name
string  rename_file
```

ARGUMENTS

-lib_name "*<lib_name>*"

Specifies the name of the Milkyway library which contains the layout cells. If not specified, write_gds will convert the cells in current opened library.

-layer_mapping "*<layer_mapping_file_name>*"

Specifies the layer mapping file name. If layer mapping file is not specified, write_gds will keep the layer information defined in the Milkyway database.

-child_depth "*<child_cell_depth>*"

Specifies the level to which you want to convert child cells. If you want to export all the cells, enter a large number, such as 20, to ensure extraction of all child cells. Milkyway exports the .CEL version of each child cell.

-output_cell "*<cell_name>*"

Specifies the cell name of the cell you want to output. By default, write_gds will output all the cells in the specified library.

-skip_ref_lib_cells

Turn this option on to skip converting child cells in reference libraries. By default, write_gds will convert all the child cells needed in reference libraries.

-text_factor "*<factor_number>*"

Specifies the text conversion factor when writing gds out. The default value is 1.0 .

-text_width "*<width_number>*"

Specifies the text width number when writing gds out. The default value is 1.0 .

-flatten_contact

Turn this option on to break contact objects down into component parts. By default, write_gds will translates contacts as references to the technology file.

-flatten_contact_array

Turn this option on to break contact array objects down into component parts. By default, write_gds will creates cells with the following naming convention from a device array: "prefix_contactName_xViaPitch_yViaPitch_xReps_yReps". Where, The "prefix" specifies the prefix you specified in the "-contact_prefix". The "contactName" specifies the name assigned to the contact in the ContactCode section of your technology file. The "xViaPitch" specifies the horizontal distance (in database units) between centers of contacts in the array. The "yViaPitch" specifies the vertical distance (in database units) between centers of contacts in the array. The "xReps" specifies the number of columns (contacts in the horizontal direction) in the array. The "yReps" specifies the number of rows (contacts in the vertical direction) in the array.

-contact_prefix "*<contact_cell_prefix>*"

Specifies the prefix you want to add to the contact or contact array names. By default, "$$" will be used.

-notch

Turn this option on to output the notch data stored in the .NOTC view. By default, notch data will not be output.

-gap

Turn this option on to output the gap data stored in the .GAP view. By default, gap data will not be output.

-fill

Turn this option on to output the fill data stored in the .FILL view. By default, fill data will not be output.

-force_output_outdate_fill

Turn this option on to force outputing fill data even the fill data is out of date.

-generate_instance_name_as_prop "*<prop_number>*"

Specifies the property number (1-127) if you want write_gds to attach a property specifying the instance name to each cell instance in the design.

-generate_geom_property

Turn this option on to output a geometry's property. By default, the property attached to a geometry will not be written out.

-strip_backslash_from_instance_net_names

Turn this option on to remove backslashes from hierarchical cell instance and net names which might conflict with hierarchical names.

-skip_layer_mapping_for_imported_cells

Turn this option on to ignore layer and data type mapping for those cells created from the GDS file during the read_gds process. The layer number and data type are written to the result GDS file without any conversion.

-output_pin_as_text

Turn this option on to output each pin's name as a TEXT object associated with pin. By default, pin will not be output.

-output_pin_as_geometry

Turn this option on to output each pin's geometry as a POLYGON object associated with pin. By default, pin will not be output.

-pin_name_mag "*<mag_number>*"

Specifies the magnification when outputting pin as TEXT object. By default, the value is 1.0 .

-rotate_pin_text_by_access_dir

Turn this option on to rotate the pins' TEXT according to the pins access direction.

-output_net_as_text

Turn this option on to output each net's name as a TEXT object associated with net. By default, it will not be output.

-output_net_as_property "*<prop_number>*"

Specifies the property number(1-127) to generate a property for each object associated with a net.

-net_name_mag "*<mag_number>*"

Specifies the magnification when outputting net as TEXT object. By default, the value is 1.0 .

-output_net_as_plex

Turn this option on to output each net as a plex number.

-long_name_length "*<name_length_limit>*"

Specifies name length limit for all the cells outputted. If the cell name exceeds the length limit, write_gds will truncate the cell name automatically. By default, the limit is 16.

-keep_data_type

Turn this option on to keep each object's data type defined in Milkyway. By default, write_gds will convert the data type to 0.

-output_odd_pin

Turn this option on to output odd shape pins. By default, odd shape pins will not be outputted.

-rename_cell_name

Specifies the file that contains the mapping from original cell names to new cell names.

The name mapping file format is

```
OldCellName NewCellName
```

For example, to map TOP.CEL to NEWTOP.CEL, enter

```
TOP NEWTOP
```

By default, the original cell names are output.

-compressed

Turn this option on to output gzip-compressed GDSII file.

<gds_file_name>

Specifies the name of the file to which to write the library data. This is a required argument.

DESCRIPTION

The **write_gds** command writes the library data of the specified library to a file in GDS format.

EXAMPLES

The following example writes out a file "*out.gds*" from the existing library "*design*".

```
write_gds -lib_name design out.gds
```

SEE ALSO

read_gds(2)

write_lef

Write the library data of the specified library to a file in LEF format.

SYNTAX

```
int write_lef
   [ -lib_name <lib_name> ]
   [ -output_cell <cell_name> ]
   [ -output_version <version> ]
   [ -ignore_tech_info ]
   [ -ignore_tech_signalEM ]
   [ -ignore_tech_antennaRule ]
   [ -ignore_tech_otherInfo ]
   [ -ignore_cell_info ]
   [ -ignore_cell_geom ]
   [ -output_via_blockages ]
   [ -output_metal_blockages ]
   [ -output_route_guides ]
   <lef_file_name>

   string lib_name
   string cell_name
   string version
```

ARGUMENTS

-lib_name *<lib_name>*

Specifies the Milkyway library that contains the design cell. The string may include the path of the library, otherwise the library is resolved from the current working directory.

-output_cell *<cell_name>*

Specifies the name of the design cell. The physical layout information of its reference cells (Milkyway standard, macro, I/O pad, corner pad, and cover cells) are written to the macro section of the LEF file.

-output_version *<version>*

Spefifies the version number for lef out. Currently, write_lef can support version 5.3|5.4|5.5|5.6|5.7|5.8. Default version is 5.5.

-ignore_tech_info

Specifies if technology information outputs to lef file.

-ignore_tech_signalEM

When ignore_tech_info is false,specifies if signalEM output to lef file.

-ignore_tech_antennaRule

When ignore_tech_info is false,specifies if antenna rule output to lef file.

-ignore_tech_otherInfo

When ignore_tech_info is false,specifies if other technology information to lef file.

-ignore_cell_info

Specifies if cell information outputs to lef file.

-ignore_cell_geom

When ignore_cell_info is false, specifies if the cell geometries output to lef file.

-output_via_blockages

When ignore_cell_info is false, specifies if the geomeries in viablockage layers output to lef file.

-output_metal_blockages

When ignore_cell_info is false, specifies if the geomeries in metalblockage layers output to lef file.

-output_route_guides

When ignore_cell_info is false, specifies if the geomeries in route guides output to lef file.

<lef_file_name>

Specifies the name of the LEF file to which the library information is written. If the file already exists, it is overwritten. If the file does not exist, it will be created. The string may include the path of the output file, otherwise the file is written to the current working directory.

DESCRIPTION

The **write_lef** command writes the library data of the specified library to a file in LEF format.

EXAMPLES

The following example writes out a file "*out.lef*" from the existing library "*design*".

```
write_lef -lib_name design out.lef
```

SEE ALSO

read_lef(2)
auNLoApi(2)

write_mw_lib_files

Writes the technology, or plib, or reference control file of the Milkyway library.

SYNTAX

```
status_value write_mw_lib_files
   [-technology]
   [-reference_control_file]
   -output file_name
   libName
```

Data Types

```
   file_name  string
   libName    string
```

ARGUMENTS

libName

Specifies the Milkyway library to be reported.

-technology

Indicates to dump technology information.

This option and the **-plib** option, the **-reference_control_file** option are mutually exclusive.

-reference_control_file

Indicates to dump the reference control file.

This option and the **-technology** option, the **-plib** option are mutually exclusive.

-output *file_name*

Specifies the file name in which the technology information or the reference library information is stored.

DESCRIPTION

Dumps the technology information or the reference library information to an ASCII file that you can edit.

At least one of the **-technology**, or **-plib**, or the **-reference_control_file** options must be specified.

A status indicating success or failure is returned.

EXAMPLES

The following example displays unit information about a Milkyway library:

```
prompt> write_mw_lib_files -reference_control_file -output ref.out design
1
```

SEE ALSO

```
create_mw_lib(2)
set_mw_lib_reference(2)
set_mw_technology_file(2)
```

write_oasis

Writes the library data of the specified library to a file in OASIS format.

SYNTAX

```
int write_oasis
   [ -lib_name <lib_name> ]
   [ -layer_mapping <layer_mapping_file_name> ]
   [ -child_depth <child_cell_depth> ]
   [ -output_cell <cell_name> ]
   [ -skip_ref_lib_cells ]
   [ -oasis_compression_level <compress_level_value> ]
   [ -flatten_contact ]
   [ -flatten_contact_array ]
   [ -contact_prefix <contact_cell_prefix> ]
   [ -notch ]
   [ -gap ]
   [ -fill ]
   [ -force_output_outdate_fill ]
   [ -generate_instance_name_as_prop <prop_number> ]
   [ -strip_backslash_from_instance_net_names ]
   [ -skip_layer_mapping_for_imported_cells ]
   [ -output_pin_as_text ]
   [ -output_pin_as_geometry ]
   [ -output_net_as_text ]
   [ -output_net_as_property <prop_number> ]
   [ -keep_data_type]
   [ -long_name_length <name_len>]
   [ -generate_geom_property]
   [ -by_layer_number]
   <oasis_file_name>

string  lib_name
string  layer_mapping_file_name
string  contact_cell_prefix
string  oasis_file_name
```

ARGUMENTS

-lib_name *<lib_name>*

Specifies the name of the Milkyway library which contains the layout cells. If not specified, write_oasis will convert the cells in current opened library.

-layer_mapping *<layer_mapping_file_name>*

Specifies the layer mapping file name. If layer mapping file is not specified, write_oasis will keep the layer information defined in the Milkyway database.

-child_depth *<child_cell_depth>*

Specifies the level to which you want to convert child cells. If you want to export all the cells, enter a large number, such as 20, to ensure extraction of all child cells. Milkyway exports the .CEL version of each child cell.

-output_cell *<cell_name>*

Specifies the cell name of the cell you want to output. By default, write_oasis will output all the cells in the specified library.

-skip_ref_lib_cells

Turn this option on to skip converting child cells in reference libraries. By default, write_oasis will convert all the child cells needed in reference libraries.

-oasis_compression_level

Specifies the compress level integer (1-9) to compress the result stream file.If this option is not specified, write_stream will generate the result stream file without compression.

-flatten_contact

Turn this option on to break contact objects down into component parts. By default, write_oasis will translates contacts as references to the technology file.

-flatten_contact_array

Turn this option on to break contact array objects down into component parts. By default, write_oasis will creates cells with the following naming convention from a device array: "prefix_contactName_xViaPitch_yViaPitch_xReps_yReps". Where, The "prefix" specifies the prefix you specified in the "-contact_prefix". The "contactName" specifies the name assigned to the contact in the ContactCode section of your technology file. The "xViaPitch" specifies the horizontal distance (in database units) between centers of contacts in the array. The "yViaPitch" specifies the vertical distance (in database units) between centers of contacts in the array. The "xReps" specifies the number of columns (contacts in the horizontal direction) in the array. The "yReps" specifies the number of rows (contacts in the vertical direction) in the array.

-contact_prefix *<contact_cell_prefix>*

Specifies the prefix you want to add to the contact or contact array names. By default, "$$" will be used.

-notch

Turn this option on to output the notch data stored in the .NOTC view. By default, notch data will not be output.

-gap

Turn this option on to output the gap data stored in the .GAP view. By default, gap data will not be output.

-fill

Turn this option on to output the fill data stored in the .FILL view. By default, fill data will not be output.

-force_output_outdate_fill

Turn this option on to force outputing fill data even the fill data is out of date.

-generate_instance_name_as_prop *<prop_number>*

Specifies the property number (1-127) if you want write_oasis to attach a property specifying the instance name to each cell instance in the design.

-strip_backslash_from_instance_net_names

Turn this option on to remove backslashes from hierarchical cell instance and net names which might conflict with hierarchical names.

-skip_layer_mapping_for_imported_cells

Turn this option on to ignore layer and data type mapping for those cells created from the OASIS file during the read_oasis process. The layer number and data type are written to the result OASIS file without any conversion.

-output_pin_as_text

Turn this option on to output each pin's name as a TEXT object associated with pin. By default, pin will not be output.

-output_pin_as_geometry

Turn this option on to output each pin's geometry as a POLYGON object associated with pin. By default, pin will not be output.

-output_net_as_text

Turn this option on to output each net's name as a TEXT object associated with net. By default, it will not be output.

-output_net_as_property *<prop_number>*

Specifies the property number(1-127) to generate a property for each object associated with a net.

-keep_data_type

Turn this option on to keep each object's data type defined in Milkyway. By default, write_oasis will convert the data type to 0.

-long_name_length *<name_length_limit>*

Specifies name length limit for all the cells outputted. If the cell name exceeds the length limit, write_oasis will truncate the cell name automatically. By default, the limit is 16.

-generate_geom_property

Turn this option on to output a geometry's property. By default, the property attached to a geometry will not be written out.

-by_layer_number *<layer_number_string>*

Specifies layer_number_string to only output the objects on the specified layers. Different layers are separated by a space character.

<oasis_file_name>

Specifies the name of the file to which to write the library data. This is a required argument.

DESCRIPTION

The **write_oasis** command writes the library data of the specified library to a file in OASIS format.

EXAMPLES

The following example writes out a file "*out.oasis*" from the existing library "*design*".

```
write_oasis -lib_name design out.oasis
```

SEE ALSO

read_oasis(2)

write_verilog

Outputs a hierarchical Verilog file.

SYNTAX

```
status write_verilog
   [-pg]
   [-split_bus]
   [-empty_module]
   [-no_physical_only_cells]
   [-no_pg_pin_only_cells]
   [-no_corner_pad_cells]
   [-no_pad_filler_cells]
   [-no_core_filler_cells]
   [-no_flip_chip_bump_cells]
   [-no_cover_cells]
   [-no_chip_cells]
   [-no_io_pad_cells]
   [-no_tap_cells]
   [-no_unconnected_cells]
   [-unconnected_ports]
   [-keep_backslash_before_hiersep]
   [-diode_ports]
   [-wire_declaration]
   [-output_net_name_for_tie]
   [-macro_definition]
   [-force_output_references ref_names_to_force]
   [-force_no_output_references ref_names_not_to_force]
   [-top_only]
   [-verbose]
   verilog_file_name
```

Data Types

```
   ref_names_to_force         string
   ref_names_not_to_force     string
```

ARGUMENTS

-pg

Generates power and ground nets and ports for all cell instances and module instances. By default, the power and ground ports are not written out.

-split_bus

Generates all bus (vector) ports as individual bits. The port names are treated as scalar port names and are escaped. For example, A module definition with the following vector port:

```
module top (out, in);
  output [3:0] out;
  input in;
  ...
endmodule
```

is generated as follows:

```
module top(\out[3] , \out[2] , \out[1] , \out[0] , in);
  output \out[3] ;
  output \out[2] ;
  output \out[1] ;
  input \out[0] ;
  output in;
  ...
endmodule
```

-empty_module

Generates empty module definitions for leaf level cells such as standard cells and macro cells. These module definitions contain port definitions but no internal module instances. By default, no module definitions are generated for leaf level cells.

-no_physical_only_cells

Prevents the writing of physical-only cell instances. When this option is on, the behavior of the command is the same as when the following options are on:

```
-no_pg_pin_only_cells
-no_core_filler_cells
```

-no_pg_pin_only_cells

Prevents the writing of cell instances that do not fall in the other 5 subtypes of physical-only and contain only a power and ground pin.

-no_corner_pad_cells

Prevents the writing of corner pad cell instances.

-no_pad_filler_cells

Prevents the writing of pad filler cell instances.

-no_core_filler_cells

Prevents the writing of core filler cell instances.

-no_cover_cells

Prevents the writing of cover cell instances.

-no_chip_cells

Prevents the writing of chip cell instances.

-no_tap_cells

Prevents the writing of tap cell instances.

-no_io_pad_cells

Prevents the writing of IO pad cell instances.

-no_flip_chip_bump_cells

Prevents the writing of flip chip bump cell instances.

-no_unconnected_cells

Prevents the writing of unconnected cell instances. Unconnected cell instances are instances with no ports connected except for power and ground ports.

-unconnected_ports

Generates unconnected ports on module instances, macro instances, and standard cell instances. All unconnected ports are generated with a "SYNOPSYS_UNCONNECTED_%d" net. The "%d" portion of the name represents the unconnected net number, which is incrementally increased every time a net is generated for uniqueness. By default, the unconnected_ports are ignored.

-keep_backslash_before_hiersep

Keeps the backslash character preceding a hierarchy separator for all identified instances. By default, the backslash character (\) is stripped.

-diode_ports

Generates diode ports for cell instances. By default, diode ports are ignored.

-wire_declaration

Generates wire declarations in the hierarchical Verilog description. This option controls only scalar wires. For vector wires, the command generates a declaration with vector limits regardless of the use of this option (switch). By default, the command generates a wire declaration only for vectors. For example, using the **-wire_declaration** option:

```
wire n246;
wire [1:0] n245;
BUF U12 (.A(n245[0]), .Y(n246));
```

Without using the **-wire_declaration** option:

```
wire [1:0] n245;
BUF U12 (.A(n245[0]), .Y(n246));
```

-output_net_name_for_tie

Generates an indirect tie as its net name, and generates a direct tie with net name SNPS_LOGIC1(0) as 1'b1(0). Otherwise, the tool generates an indirect tie as 1'b1(0), as well as a direct tie.

-macro_definition

Generates a module definition for a soft macro. By default, it is not generated. A soft macro is a user-partitioned macro, generated by your design planning steps. Most of time, big designs tend to be divided into several partitioned macros. By default, a soft macro is treated as a leaf macro and its definition is not output. When you use this option, the Verilog writer looks inside the soft macro and writes out the internal details; and it can also write the full contents of an interface logic module (ILM) block in the verilog netlist.

-force_output_references ***ref_names_to_force***

Specifies reference cell names, separated with a space, for which all cell instances must be generated overriding the following options:

```
-no_physical_only_cells
-no_pg_pin_only_cells
-no_corner_pad_cells
-no_pad_filler_cells
-no_core_filler_cells
-no_flip_chip_bump_cells
-no_cover_cells
-no_chip_cells
-no_io_pad_cells
-no_tap_cells
-no_unconnected_cells
```

The maximum allowed string size is 1023 chars.

-force_no_output_references ***ref_names_not_to_force***

Specifies reference cell names, separated with a space, for which all cell instances must *not* be generated, overriding the same options as listed in the description for the **-force_output_references** option. For example, consider a cell named "top" with the following 4 types of core filler cells FILL1, FILL2, FILL4, and FILL8. To generate the Verilog description with only the FILL4 and FILL8 instances, use the **-no_core_filler_cells** option and the **-force_output_references** option with the value FILL4 FILL8 for the *ref_names_to_force* argument.

-top_only

Generate top module only. By default, the whole hierarchical netlist is output, not only the top module.

-verbose

Output more informations such as warning messages.

verilog_file_name

Specifies the name of the file to which the hierarchical Verilog description is written.

DESCRIPTION

This command generates a Verilog netlist for a given Milkyway design. By specifying various options, you can control the behavior of this command.

EXAMPLES

The following example shows the most common use of this command.

```
prompt> write_verilog top_hvo.v
Generating description for top level cell.
Processing module sub
Processing module top
Elapsed = 0:00:00, CPU = 0:00:00
Write verilog completed successfully.
```

SEE ALSO

read_verilog(2)