PrimeTime® Constraint Consistency 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

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]
```

Data Types

```
   base_collection    collection
   object_spec        list
```

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. The *base_collection* option 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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 (using the **get_ports** command) gets all ports beginning with mode,then adds the CLOCK port.

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

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

```
ptc_shell> set sp [get_ports SCANOUT]
{"SCANOUT"}
ptc_shell> set het [get_cells u1]
{"u1"}
ptc_shell> 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, if 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.

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

ptc_shell> 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)

ptc_shell> 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)

alias

Creates a pseudo-command that expands to one or more words, or lists current alias definitions.

SYNTAX

```
string alias
[name]
[def]
```

Data Types

name	string
def	string

ARGUMENTS

name

Specifies a name of the alias to define or display. The name must begin with a letter, and can contain letters, underscores, and numbers.

def

Expands the alias. That is, the replacement text for the alias name.

DESCRIPTION

The **alias** command defines or displays command aliases. With no arguments, the **alias** command displays all currently defined aliases and their expansions. With a single argument, the **alias** command displays the expansion for the given alias name. With more than one argument, an alias is created that is named by the first argument and expanding to the remaining arguments.

You cannot create an alias using the name of any existing command or procedure. Thus, you cannot use **alias** to redefine existing commands.

Aliases can refer to other aliases.

Aliases are only expanded when they are the first word in a command.

EXAMPLES

Although commands can be abbreviated, sometimes there is a conflict with another command. The following example shows how to use **alias** to get around the conflict:

```
prompt> alias q quit
```

The following example shows how to use **alias** to create a shortcut for commonly-used command invocations:

```
prompt> alias include {source -echo -verbose}

prompt> alias rt100 {report_timing -max_paths 100}
```

After the previous commands, the command **include script.tcl** is replaced with **source -echo -verbose script.tcl** before the command is interpreted.

The following examples show how to display aliases using **alias**. Note that when displaying all aliases, they are in alphabetical order.

```
prompt> alias rt100
rt100  report_timing -max_paths 100

prompt> alias
include  source -echo -verbose
q  quit
rt100  report_timing -max_paths 100
```

SEE ALSO

unalias(2)

all_clocks

Creates a collection of all clocks in the current design. You can assign these clocks to a variable or pass them into another command.

SYNTAX

```
collection all_clocks
```

ARGUMENTS

The **all_clocks** command has no arguments.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_clocks** command creates a collection of all clocks in the current design. If you do not define any clocks, the empty collection (empty string) is returned.

If you want only certain clocks, use **get_clocks** to create a collection of clocks matching a specific pattern and optionally pass in filter criteria.

EXAMPLES

The following example applies the **set_propagated_clock** command to all clocks in the design.

```
ptc_shell> set_propagated_clock [all_clocks]
```

SEE ALSO

collections(2)
create_clock(2)
derive_clocks(2)
get_clocks(2)
set_propagated_clock(2)

all_connected

Creates a collection of objects connected to a net, pin, or port object. You can assign this collection to a variable or pass it into another command.

SYNTAX

```
collection all_connected
   [-leaf]
   object_spec
```

Data Types

```
   object_spec     list
```

ARGUMENTS

-leaf

Specifies that the connections of the net that are being returned should be global or leaf pins. When specified, this gives the leaf pins of a hierarchical net. For nonhierarchical nets, there is no difference in output.

object_spec

Specifies the object whose connections are returned. This is a collection of one element which is a net, pin, or port collection, or the name of a net, pin, or port.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_connected** command creates a collection of objects connected to a specified net, pin, or port. The *object_spec* option is either a collection of exactly one net, pin, or port object, or a name of an object. If it is a name, constraint consistency searches for a net, pin, or port, in that order. If the *object_spec*

refers to a net, the *-leaf* option can be used to get the global leaf pins of the net.

The command returns a collection. The collection can contain nets, ports, pins, or a combination of ports and pins. In the latter case, the ports comes first, followed by the pins.

When issued from the command prompt, **all_connected** 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**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example shows all objects connected to net "CLOCK":

```
ptc_shell> query_objects -verbose [all_connected [get_nets CLOCK]]
{"port:CLOCK", "pin:U1/CP", "pin:U2/CP", "pin:U3/CP", "pin:U4/CP"}
```

SEE ALSO

collections(2)
get_nets(2)
get_pins(2)
query_objects(2)
collection_result_display_limit(3)

all_exceptions

Creates a collection of all timing exceptions in the current design.

SYNTAX

```
collection all_exceptions
   [-filter expression]
   [-quiet]
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For all exceptions, the expression is evaluated based on the exception's attributes. If the expression evaluates to true, the exception is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_exceptions** command creates a collection of all timing exceptions that pass the filter (if specified) in the current scenario. If no timing exceptions exist, the empty string is returned. You can assign these exceptions to a variable or pass them into another command. Timing exceptions are created by **set_false_path**, **set_multicycle_path**, **set_max_delay** or **set_min_delay**.

If you want only certain exceptions, use **get_exceptions** to create a collection of exceptions matching a specific pattern and optionally pass in filter criteria.

EXAMPLES

This example sets the variable allExcepts to a collection of all timing exceptions in the design.

```
ptc_shell> set allExcepts [all_exceptions]
```

SEE ALSO

collections(2)
set_false_path(2)
set_multicycle_path(2)
set_max_delay(2)
set_min_delay(2)
get_exceptions(2)
collection_result_display_limit(3)

all_fanin

Creates a collection of pins, ports, or cells in the fanin of specified sinks.

SYNTAX

```
collection all_fanin -to to_list
   [-from from_list]
   [-through through_list]
   [-startpoints_only]
   [-only_cells]
   [-flat]
   [-quiet]
   [-levels level_count]
   [-pin_levels pin_count]
   [-trace_arcs arc_types]
```

Data Types

```
   to_list       list
   from_list     list
   through_list  list
   level_count   int
```

ARGUMENTS

-to *to_list*

Specifies a list of sink pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. The timing fanin of each sink in *to_list* becomes part of the resulting collection. If a net is specified, the effect is the same as listing all driver pins on the net. This argument is required.

-through *through_list*

Specifies a list of pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. If a *through_list* is specified the fanin of each object in *to_list* is restricted to objects on paths through the pins, ports or nets in *through_list*.

-from *from_list*

Specifies a list of pins, ports, or nets in the design. Each object is a named pin, port, or net, or a

collection of pins, ports, or nets. The timing fanin of each object in *to_list* becomes part of the resulting collection only if it is in the fanout cone of at least one object in *from_list*. If a net is specified, the effect is the same as listing all load pins on the net.

-startpoints_only

Includes only the timing startpoints in the result.

-only_cells

Includes only the cells in the timing fanin of the *sink_list* and not pins or ports.

-flat

There are two major modes in which **all_fanin** functions: hierarchical (the default) and flat. In hierarchical mode, only objects within the same hierarchical level as the current sink are included in the result. In flat mode, the only non-leaf objects in the result are hierarchical sink pins.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-levels *cell_count*

The traversal stops when reaching a depth of search of *cell_count* hops, where the counting is performed over the layers of cells of the same distance from the sink.

-pin_levels *pin_count*

The traversal stops when reaching a depth of search of *pin_count* hops, where the counting is performed over the layers of pins of the same distance from the sink.

-trace_arcs *arc_types*

Specifies the type of combinational arcs to trace during the traversal. Allowed values are

- **timing** (the default) - Permits tracing only of valid timing arcs -- that is, arcs that are neither disabled nor invalid due to case analysis.
- **enabled** - Permits the tracing of all enabled arcs and disregards case analysis values.
- **all** - Permits the tracing of all combinational arcs regardless of either case analysis or arc disabling.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_fanin** command creates a collection of objects in the timing fanin of specified sink pins/ports or nets in the design. A pin is considered to be in the timing fanin of a sink if there is a timing path through combinational logic from the pin to that sink (see also the **-trace_arcs** option). The fanin stops at the clock pins of registers (sequential cells).

If a current instance in the design is not the top level of hierarchy, only objects within the current instance are returned.

EXAMPLES

This example shows the timing fanin of a port in the design. The fanin includes a register, reg1.

```
ptc_shell> query_objects [all_fanin -to out_1]
{"out_1", "reg1/Q", "reg1/CP"}
```

This example shows the flat mode of the **all_fanin** command. The sink is an input pin of a hierarchical cell, H1, which is connected to an output pin of another hierarchical cell, H2. H2 contains additional hierarchy and eventually, a leaf cell with 2 inputs, each of which has a top-level register in its fanin.

```
ptc_shell> query_objects [all_fanin -to H1/a -flat]
{"H1/a", "H2/U1/n1/Z", "H2/U1/n1/A", "H2/U1/n1/B",
 "reg1/Q", "reg2/Q", "reg1/CP", "reg2/CP"}
```

SEE ALSO

all_fanout(2)
current_instance(2)
report_transitive_fanin(2)
report_transitive_fanout(2)

all_fanout

Creates a collection of pins, ports, or cells in the fanout of the specified sources.

SYNTAX

```
collection all_fanout
   -from from_list
   -clock_tree
   [-through through_list]
   [-to to_list]
   [-flat]
   [-quiet]
   [-endpoints_only]
   [-only_cells]
   [-levels level_count]
   [-pin_levels pin_count]
   [-trace_arcs arc_types]
```

Data Types

```
   from_list        list
   to_list          list
   through_list     list
   level_count      int
```

ARGUMENTS

-from *from_list*

Specifies a list of source pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. The timing fanout of each source in *from_list* becomes part of the resulting collection. If a net is specified, the effect is the same as listing all of the load pins on the net. This option is exclusive with the **-clock_tree** option.

-through *through_list*

Specifies a list of pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. If a *through_list* is specified the fanout of each object in *from_list* is restricted to pins on paths through the pins, ports or nets in *through_list*.

-to *to_list*

Specifies a list of pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. The timing fanout of each object in *from_list* becomes part of the resulting collection if it is in the fanin cone of at least one object in *to_list*. If a net is specified, the effect is the same as listing all driver pins on the net.

-clock_tree

Indicates that all of the clock source pins and ports in the design are to be used as the list of sources. Clock sources are specified using the **create_clock** option. If there are no clocks, or if the clocks have no sources, the result is the empty collection. This option is exclusive with the **-from** option.

-flat

There are two major modes in which **all_fanout** functions: hierarchical (the default) and flat. In hierarchical mode, only objects within the same hierarchical level as the current source are included in the result. In flat mode, only non-leaf objects in the result are hierarchical source pins.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-endpoints_only

Includes only the timing endpoints in the result.

-only_cells

Includes only the cells in the timing fanout of the *source_list* and not pins or ports in the result.

-levels ***cell_count***

Specifies that the traversal stops when reaching a depth of search of *cell_count* hops, where the counting is performed over the layers of cells of the same distance from the source.

-pin_levels ***pin_count***

Specifies that the traversal stops when reaching a depth of search of *pin_count* hops, where the counting is performed over the layers of pins of the same distance from the source.

-trace_arcs ***arc_types***

Specifies the type of combinational arcs to trace during the traversal. Allowed values are

- **timing** (the default) - Permits tracing only of valid timing arcs -- that is, arcs that are neither disabled nor invalid due to case analysis.
- **enabled** - Permits the tracing of all enabled arcs and disregards case analysis values.
- **all** - Permits the tracing of all combinational arcs regardless of either case analysis or arc disabling.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_fanout** command creates a collection of objects in the timing fanout of specified source pins, ports, or nets in the design. A pin is considered to be in the timing fanout of a source if there is a timing path through combinational logic from that source to the pin (see also the **-trace_arcs** option). The fanout stops at the inputs to registers (sequential cells). The sources are specified using either the **-clock_tree** or **-from** *source_list* options.

If the current instance in the design is not the top level of hierarchy, only objects within the current instance are returned.

EXAMPLES

This example shows the timing fanout of a port in the design. The fanout includes a register, reg3.

```
ptc_shell> query_objects [all_fanout -from in1]
{"in1", "reg3/D"}
```

This example shows the difference between the hierarchical and flat modes of the **all_fanout** command. The source is an output pin of a hierarchical cell, H3/z1, which is connected to an input pin of another hierarchical cell, H4/a. H4 contains a leaf cell U1 with input A and output Z. The first command is in hierarchical mode, and shows that hierarchical pins are included in the result. The second command is in leaf mode, and leaf pins from the lower level are included.

```
ptc_shell> query_objects [all_fanout -from H3/z1]
{"H3/z1", "H4/a", "H4/z", "reg2/D"}

ptc_shell> query_objects [all_fanout -from H3/z1 -flat]
{"H3/z1", "H4/U1/A", "H4/U1/Z", "reg2/D"}
```

SEE ALSO

```
all_fanin(2)
current_instance(2)
report_transitive_fanin(2)
report_transitive_fanout(2)
```

all_inputs

Creates a collection of all input ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_inputs
   [-level_sensitive]
   [-edge_triggered]
   [-clock clock_name]
   [-exclude_clock_ports]
```

Data Types

```
   clock_name     list
```

ARGUMENTS

-level_sensitive

Considers only those ports with level-sensitive input delay. This is specified by the **set_input_delay 2 -clock CLK -level_sensitive IN1** command.

-edge_triggered

Considers only those ports with edge-triggered input delay. This is specified by the **set_input_delay 2 -clock CLK IN2** command.

-clock *clock_name*

Considers only those ports with input delay relative to a specific clock. This can be the name of a clock, or a collection containing a clock.

-exclude_clock_ports

Do not consider clock ports. For example, ports on which a clock is defined will not be reported.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_inputs** command creates a collection of all input or inout ports in the current design. You can limit the contents of the collection by specifying the type of input delay that must be on a port.

If you want only certain ports, use the **get_ports** command to create a collection of ports matching a specific pattern and optionally passing filter criteria.

When issued from the command prompt, the **all_inputs** command behaves as though the **query_objects** command 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 **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example specifies a driving cell for all input ports.

```
ptc_shell> set_driving_cell -lib_cell FFD3 -pin Q [all_inputs]
```

SEE ALSO

```
collections(2)
get_ports(2)
report_port(2)
query_objects(2)
set_driving_cell(2)
set_input_delay(2)
collection_result_display_limit(3)
```

all_instances

Creates a collection of all instances of a specific design or library cell in the current design, relative to the current instance. You can assign the resulting collection of cells to a variable or pass it into another command.

SYNTAX

```
collection all_instances
   [-hierarchy]
   object_spec
```

Data Types

```
   object_spec     string
```

ARGUMENTS

-hierarchy

Searches for instances in all levels of instance hierarchy below the current instance. By default, only instances from the current level of hierarchy are considered.

object_spec

Specifies the target design or library cell. This can be a design collection, lib_cell collection, or name.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_instances** command creates a collection of cells that are instances of a design or library cell. The search for instances is made relative to the current instance within the current design. By default, the **all_instances** command considers only instances at the current level of the hierarchy. If you use the **-hierarchy** option, the search continues throughout the hierarchy.

The *object_spec* can be a simple name. In this case, any instance of a design or lib_cell with that name will match. Alternatively, the *object_spec* can be a collection of exactly one design or one library cell. Any other collection results in an error message. Using the collection can help focus the search for instances of specific designs or lib_cells, especially after **swap_cell** has been used.

When issued from the command prompt, the **all_instances** command 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.

EXAMPLES

The following example uses this command to get the instances of the design, "low" in the current level of hierarchy.

```
ptc_shell> all_instances low
{"U1", "U3"}
```

The following example uses **-hierarchy** to display instances of a design across multiple levels of hierarchy.

```
ptc_shell> query_objects [all_instances low -hierarchy]
{"U1", "U2/U1", "U3"}
```

In the following example, there are two instances of design, "inter". One of them is swapped for a new design. The difference between using **all_instances** with a name and a collection of one design is shown.

```
ptc_shell> swap_cell i1 inter_2.db:inter
...unlinking i1
1
ptc_shell> all_instances inter
{"i1", "i2"}
ptc_shell> all_instances [get_designs inter_2.db:inter]
{"i1"}
```

SEE ALSO

current_design(2)
current_instance(2)
get_designs(2)
get_lib_cells(2)
get_cells(2)
query_objects(2)
collection_result_display_limit(3)

all_outputs

Creates a collection of all output ports in the curr8ent design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_outputs [-level_sensitive]
   [-level_sensitive]
   [-edge_triggered]
   [-clock clock_name]
```

Data Types

```
   clock_name     list
```

ARGUMENTS

-level_sensitive

Considers only the ports with level-sensitive output delay. This is specified by **set_output_delay 2 -clock CLK -level_sensitive IN1**.

-edge_triggered

Considers only the ports with edge-triggered output delay. This is specified by **set_output_delay 2 -clock CLK IN2**.

-clock *clock_name*

Considers only the ports with output delay relative to a specific clock. This can be the name of a clock, or a collection containing a clock.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_outputs** command creates a collection of all output or inout ports in the current design. You can limit the contents of the collection by specifying the type of output delay that must be on a port.

If you want only certain ports, use the **get_ports** command to create a collection of ports that match a specific pattern, and optionally, pass the filter criteria.

When issued from the command prompt, the **all_outputs** command behaves as though the **query_objects** command is called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change the maximum number of objects displayed 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 specifies a pin capacitance for all output ports.

```
ptc_shell> set_load 4.56 [all_outputs]
```

The following example shows the names of all of the output ports with the output delay relative to PHI1.

```
ptc_shell> all_outputs -clock PHI1
```

SEE ALSO

```
collections(2)
get_ports(2)
report_port(2)
query_objects(2)
set_output_delay(2)
collection_result_display_limit(3)
```

all_registers

Creates a collection of register cells or pins. You can assign the resulting collection to a variable or pass it into another command.

SYNTAX

```
collection all_registers
   [-clock clock_name]
   [-rise_clock rise_clock_name]
   [-fall_clock fall_clock_name]
   [-cells]
   [-data_pins]
   [-clock_pins]
   [-slave_clock_pins]
   [-async_pins]
   [-output_pins]
   [-level_sensitive]
   [-edge_triggered]
   [-master_slave]
   [-no_hierarchy]
```

Data Types

```
   clock_name              list
   rise_clock_name         list
   fall_clock_name         list
```

ARGUMENTS

-clock *clock_name*

Considers all registers clocked by the *clock_name* argument. This is either the name of a clock, or a collection containing a clock. For example, all registers whose clock pins are in the fan out of the specified clock are considered.

-rise_clock *rise_clock_name*

Considers all registers clocked by the *rise_clock_name* argument and having an open edge, effectively the rising clock edge. This is either the name of a clock, or a collection containing a clock. For example, rising edge-triggered flip-flops without any inversion on the clock path or falling edge-triggered flip-flops with inversion on the clock path are considered.

-fall_clock *fall_clock_name*

Considers all registers clocked by the *fall_clock_name* argument and having an open edge, effectively falling the clock edge. This is either the name of a clock, or a collection containing a clock. For example, rising edge-triggered flip-flops with inversion on the clock path or falling edge-triggered flip-flops without any inversion on the clock path are considered.

-cells

Creates a collection of cells (default). The cells are registers and are further limited by other command options.

-data_pins

Creates a collection of register data pins. The collection can be limited by other command options.

-clock_pins

Creates a collection of register clock pins. The collection can be limited by other command options.

-slave_clock_pins

Creates a collection of register slave clock pins (the slave clock pins of master-slave registers). Specify slave clock pins as **clocked_on_also** in the library.

-async_pins

Creates a collection of asynchronous preset or clear pins.

-output_pins

Creates a collection of register output pins.

-level_sensitive

Limits search to level-sensitive latches.

-edge_triggered

Limits search to edge-triggered flip-flops.

-master_slave

Only considers master or slave register cells.

-no_hierarchy

Only searches the current instance; does not descend the hierarchy.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_registers** command creates a collection of pins or cells related to registers.

When issued from the command prompt, this command behaves as though the **query_objects** command is called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum number of objects displayed by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

SEE ALSO

```
collections(2)
get_cells(2)
get_pins(2)
collection_result_display_limit(3)
```

all_scenarios

Creates a collection of all scenarios in the current design. You can assign these scenarios to a variable or pass them into another command.

SYNTAX

```
collection all_scenarios
```

ARGUMENTS

The **all_scenarios** command has no arguments.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **all_scenarios** command creates a collection of all of the scenarios in the current design. If you do not define any scenarios, a collection containing the default scenario is returned.

If you want only certain scenarios, use the **get_scenarios** command to create a collection of scenarios that match a specific pattern and optionally pass in filter criteria.

EXAMPLES

This example passes a collection of all scenarios to the **analyze_design** command.

```
ptc_shell> analyze_design -scenarios [all_scenarios]
```

SEE ALSO

```
collections(2)
create_scenario(2)
get_scenarios(2)
```

analyze_clock_networks

Performs an analysis of all of the paths satisfying the specification.

SYNTAX

```
Boolean analyze_clock_networks
   -from from_list | -through through_list | -to to_list
   [-from from_list]
   [-through through_list]*
   [-to to_list]
   [-end_types end_type_list]
   [-max_endpoints limit]
   [-style style]
   [-traverse_disabled]
   [-nosplit]
   [-text_only]
```

Data Types

```
   from_list         list
   through_list      list
   to_list           list
   end_type_list     list
   limit             integer
   format            string
```

ARGUMENTS

-from *from_list*

Specifies a list of clocks (except for virtual clocks), clock source ports, or clock source pins that the selected clock network is to begin. If a cell is specified, the analysis starts at any clock source defined on a pin of that cell. If the **from_list** does not include any valid clock source objects, an error occurs. This option is required if neither the **-through** or the **-to** option is specified.

-through *through_list*

Specifies a list of pins, ports, cells, and nets through which the clock networks must pass. Nets are interpreted to imply the leaf-level driver pins.

You can specify the **-through** option more than one time in a single command invocation. If none of the objects in the **through_list** are part of a clock network, or if none of the clock networks match

those for any **from_list** or **to_list**, an error occurs. This option is required if neither the **-from** or the **-to** option is specified.

-to ***to_list***

Specifies a list of clock network endpoint objects. The types of endpoints considered is specified by the **-end_types** option. This option is required if neither the **-from** or the **-through** option is specified.

-end_types ***end_type_list***

Lists the types of clock network endpoints to report. Possible values of endpoint types are: **register**, **port**, **clock_source**, **data**, **clock_gating**, **async_pin**, **internal_end**.

Clock network endpoints of the **internal_end** type are located where a branch of the clock network ends, as there is no logic forward from that location. The **internal_end** endpoints are caused by issues such as unconnected output pins or unresolved cell references.

The default for **end_type_list** is {register port clock_source}.

-max_endpoints

Reduces the paths displayed to at most the given number of outputs.

-style ***style***

Specifies the style of report to print. Possible values for the **-style** are: **short**, **end**, **summary**, **full**, **full_expanded**, **summary_expanded**. The default is **short**. A description and example of the different styles is provided in the EXAMPLE section.

-traverse_disabled

Causes the analysis to include a clock network that is disabled due to constraints such as case, **set_disable_timing**, and **set_max_delay**. This corresponds to the potential network that the **potential_clocks** attribute shows.

-nosplit

Prohibits the breaking of lines in the reports when columns overflow.

-text_only

Does not bring up the clock network schematic in GUI mode if the style is **full** or **full_expanded**. It has no effect if the given style is other than **full** or **full_expanded**.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command allows you to explore clock networks. There are six styles for reporting the clock network:

- **end** - Reports clock pins and other clock network endpoints. The report includes which clocks reach that endpoint and optionally which clocks where blocked from reaching those endpoint.

- **short** - Like the end report, but only unique clock, potential clock, and blocking constraint combinations are displayed.
- **full** - Reports each pin in the clock networks that satisfy the **-from -through -to** specification given. The report shows the pins in paths and partial paths. The partial paths are referred to as branches. A schematic window containing the clock network is brought up if in GUI mode, unless **-text_only** option is given.
- **summary** - Like the full report, but the report is compressed to only show pins where "something interesting" happens. Pins that do not have a change in clock sense, start or end a partial path branch are not included.
- **full_expanded** - Like the full report, but the paths start at the primary master clock sources and proceed through all generated clocks to the clock network endpoints.
- **summary_expanded** - Like the full_expanded report, but pins that do not have a change in clock sense or start or end a partial path are not included.

EXAMPLES

-to clock_pin To view all clocks for a clock pin, use the **-to** option. The short or end report shows the sets of clocks that arrive at the endpoint.

```
ptc_shell> analyze_clock_network -to blk1/ff1a/CP
**************************************
Report : analyze_clock_networks
         -max_endpoints=1000
         -style short
         -end_types  {register port clock_source }
Design : blockG
Version: ...
Date   : Wed Feb 13 16:07:24 2008
**************************************
Clock Network End Type Abbreviations:
   R = register
   P = port
   CS = clock_source
                                End
Example End Pin/Port            Type Clocks                  Count
-------------------------------------------------------------------
blk1/ff1a/CP                    R    set 0                       1

Clock Sets:
  set 0 (3 clocks):
    clk - negative
    clk - positive
    test_clock - positive
```

This example shows **-style full** with reconvergence. The following example shows how clock network reconvergence is displayed in the **-style full** report. The clock network to the port "selclkout" fans out to both inputs of a mux. The Branch Info column shows "S1" where the branch 1 partial path starts, and shows "E1" where the branch 1 partial path ends. The Sense Notes column shows that the positive unate and negative unate sense merge at the pin "mux/Z." The second report shows the same command output after the command:

```
    set_clock_sense -negative mux/Z
```

The third report shows the output with **-traverse_disabled** after the **set_clock_sense** command was issued.

```
ptc_shell> analyze_clock_networks -style full -to selclkout
*************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full
        -end_types  {register port clock_source }
Design : clock_network2
*************************************
Key for clock sense abbreviations:
   P = positive
   N = negative
   P,N = positive, negative

Full report for clock: clk - 2 partial path branches

Branch 0:
       Branch
Level  Info     Sense Notes            Port/Pin
-------------------------------------------------------------------------
    0  S1       P     source           clk
    1           P                      inv/A
    2           N                      inv/Z
    3           N                      mux/A
    4  E1       P,N                    mux/Z
    5           P,N                    both_buf/A
    6           P,N                    both_buf/Z
    7           P,N   port             selclkout

Branch 1: from branch 0 reconverges to branch 0
       Branch
Level  Info     Sense Notes            Port/Pin
-------------------------------------------------------------------------
    1           P                      in_buf/A
    2           P                      in_buf/Z
    3           P                      mux/B

ptc_shell> set_clock_sense -negative mux/Z

ptc_shell>analyze_clock_networks -style full -to selclkout
*************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full
        -end_types  {register port clock_source }
Design : clock_network2
*************************************
Key for clock sense abbreviations:
   P = positive
   N = negative

Full report for clock: clk

Branch 0:
       Branch
Level  Info     Sense Notes            Port/Pin
-------------------------------------------------------------------------
    0           P     source           clk
    1           P                      inv/A
    2           N                      inv/Z
```

```
    3           N                       mux/A
    4           N     user_sense        mux/Z
    5           N                       both_buf/A
    6           N                       both_buf/Z
    7           N     port              selclkout

ptc_shell> analyze_clock_networks -style full \
                                  -to selclkout -traverse_disabled
************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full
        -end_types  {register port clock_source }
        -traverse_disabled
Design : clock_network2
************************************
Key for clock sense abbreviations:
   P = positive
   N = negative
   Potential senses detected with -traverse_disabled:
   [P] = potential positive

Full report for clock: clk - 2 partial path branches

Branch 0:
       Branch
Level  Info     Sense Notes            Port/Pin
-------------------------------------------------------------------------
    0  S1       P     source           clk
    1           P                      inv/A
    2           N                      inv/Z
    3           N                      mux/A
    4  E1       N [P] user_sense       mux/Z
    5           N [P]                  both_buf/A
    6           N [P]                  both_buf/Z
    7           N [P] port             selclkout

Branch 1: from branch 0 reconverges to branch 0
       Branch
Level  Info     Sense Notes            Port/Pin
-------------------------------------------------------------------------
    1           P                      in_buf/A
    2           P                      in_buf/Z
    3           P                      mux/B
```

EXAMPLE: full and full_expanded Styles

The following example shows a register, blk3/ff1a/CP where the clock div4clk is blocked by a case value on buf2/Z. Use the **-traverse_disabled** option to see the paths. The clock sense is placed in square braces where it is only a potential clock sense.

```
ptc_shell> analyze_clock_network -style full \
                                 -to blk3/ff1a/CP -traverse_disabled
************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full
        -end_types  {register port clock_source }
        -traverse_disabled
Design : clock_div4
Version: ...
Date   : Thu Feb 14 11:10:54 2008
************************************
```

```
   Key for clock sense abbreviations:
      P = positive
      Potential senses detected with -traverse_disabled:
      [P] = potential positive

   Full report for clock: d4

   Branch 0:
          Branch
   Level  Info     Sense Notes            Port/Pin
   ------------------------------------------------------------------------
       0           P     source           div4/Q
       1           P                      buf2/A
       2           [P]   case             buf2/Z
       3           [P]   register, case   blk3/ff1a/CP
   1
```

The following is the same report, but the -style full_expanded was used so the paths from the master clock source to the generated clock sources are shown.

```
ptc_shell> analyze_clock_network -to blk3/ff1a/CP \
                                 -style full_expanded -traverse_disabled
****************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full_expanded
        -end_types  {register port clock_source }
        -traverse_disabled
Design : clock_div4
Version: ...
Date   : Thu Feb 14 11:13:16 2008
****************************************
Key for clock sense abbreviations:
   P = positive
   N = negative
   R = rise_triggered
   Potential senses detected with -traverse_disabled:
   [P] = potential positive

Source latency paths for generated clock: div2clk
   from master clock: clk

Branch 0:
       Branch
Level  Info     Sense Notes            Port/Pin
------------------------------------------------------------------------
    0           P     source           clk
    1           P                      inv1/A
    2           N                      inv1/Z
    3           N                      inv2/A
    4           P                      inv2/Z
    5           P     clock_gating     gate2/A
    6           P                      gate2/Z
    7           P     register         div2/CP
    8           P                      div2/CP
    9           R     clock_source     div2/Q

Source latency paths for generated clock: div4clk
   from master clock: div2clk

Branch 0:
       Branch
Level  Info     Sense Notes            Port/Pin
------------------------------------------------------------------------
```

```
    0           P     source          div2/Q
    1           P                     buf1/A
    2           P                     buf1/Z
    3           P     register        div4/CP
    4           P                     div4/CP
    5           R     clock_source    div4/Q


Full report for clock: d4

Branch 0:
       Branch
Level  Info     Sense Notes            Port/Pin
-------------------------------------------------------------------
    0           P     source          div4/Q
    1           P                     buf2/A
    2           [P]   case            buf2/Z
    3           [P]   register, case  blk3/ff1a/CP
```

"EXAMPLE: Looking at All Clock Combinations

To look at what combination of clocks arrive at clock network endpoints use the **-style short** and **-from [all_clocks] -max_endpoints 1000000**. Each unique combination of endpoint type and clock set is printed. The Count column is how many endpoints share the same end type and clock set. The Example End Pin/Port column is arbitrary.

```
ptc_shell> analyze_clock_network -from [all_clocks] -max_endpoints 1000000
**************************************
Report : analyze_clock_networks
        -max_endpoints=1000000
        -style short
        -end_types  {register port clock_source }
Design : blockC
Version: ...
Date   : ...
**************************************
Clock Network End Type Abbreviations:
   R = register
   P = port
   CS = clock_source
                                End
Example End Pin/Port            Type Clocks              Count
-------------------------------------------------------------
blk1/ff2b/CP                    R    set 0                3239
div4/CP                         R    set 1                   1
blk3/ff1a/CP                    R    set 2                 116

Clock Sets:
  set 0 (1 clocks):
    clk - positive
  set 1 (1 clocks):
    div2_clk - positive
  set 3 (1 clocks):
    div4_clk - positive
```

SEE ALSO

```
analyze_paths(2)
```

analyze_design

Performs rule checking and additional analysis of the design.

SYNTAX

```
string analyze_design
   [-scenarios scenario_list]
   [-rules rule_list]
   [-verbose]
   [-significant_digits digits]
```

Data Types

```
   scenario_list     list
   rule_list         list
   digits            list
```

ARGUMENTS

-scenarios *scenario_list*

Specifies a list of scenarios to include in the output. If not specified, all scenarios are included.

-rules *rule_list*

Specifies a list of rules or rule sets. The *analyze_design* command checks only these rules. If not specified, the *analyze_design* command checks all enabled rules.

-verbose

If specified, prints verbose status information.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time-related values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default is *2*. Use this option if you want to override the default. This argument controls only the number of digits displayed, not the precision used internally for analysis. For analysis, the tool uses the full precision of the platform's floating-point arithmetic capability.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Performs rule checking to find potential problems in the design and its constraints. Produces a violation repository that can be viewed in the GUI or via the *report_constraint_analysis* command. If user-defined rules are checked by the *analyze_design* command, the user-defined checker procedures are invoked automatically. The output of the user-defined rule check procedures is integrated into the *analyze_design* output.

EXAMPLES

The following example performs rule checking on all scenarios.

```
ptc_shell> analyze_design
```

SEE ALSO

```
create_rule(2)
set_rule_property(2)
report_default_significant_digits(2)
report_constraint_analysis(2)
```

analyze_paths

Performs an analysis of all the paths satisfying the specification.

SYNTAX

```
Boolean analyze_paths
   [-rise | -fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-path_type format]
   [-max_endpoints]
   [-traverse_disabled]
   [-unconstrained]
   [-nosplit]
   [-text_only]
   [-max_paths]
```

Data Types

```
   from_list          list
   rise_from_list     list
   fall_from_list     list
   through_list       list
   rise_through_list  list
   fall_through_list  list
   to_list            list
   rise_to_list       list
   fall_to_list       list
   format             string
   max_paths          long
```

ARGUMENTS

-rise

Indicates that only rising paths are to be analyzed. If neither the **-rise** nor **-fall** options is specified,

both rising and falling delays are analyzed. Rise refers to a rising value at the path endpoint.

-fall

Indicates that only falling path delays are to be analyzed. If neither the **-rise** nor **-fall** options is specified, both rising and falling delays are analyzed. Fall refers to a falling value at the path endpoint.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from ***rise_from_list***

Similar to the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by the rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from** options.

-fall_from ***fall_from_list***

Similar to the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-through ***through_list***

Specifies a list of pins, ports, cells, and nets through which the multicycle paths must pass. Nets are interpreted to imply the net segment's local driver pins. If you omit the **-through** option, all timing paths specified using the **-from** and **-to** options are affected. You can specify the **-through** option more than one time in a single command invocation.

-rise_through ***rise_through_list***

Similar to the **-through** option, but applies only to paths with a rising transition at the specified objects. You can specify the **-rise_through** option more than one time in a single command invocation.

-fall_through ***fall_through_list***

Similar to the **-through** option, but applies only to paths with a fall transition at the specified objects. You can specify the **-fall_through** option more than one time in a single command invocation.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-rise_to ***rise_to_list***

Similar to the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by the rising

edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-fall_to ***fall_to_list***

Similar to the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by the falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-path_type ***format***

The format can be end, summary, or full. The end format reports the paths for each specified endpoint. The summary format reports the paths between each pair of start and endpoints. The full format prints all of the pins in the paths that satisfy the given from/through/to specification. The default format is end. When the "full" format is used and the GUI is in session, he schematic for paths reported is generated.

-max_endpoints

Reduces the paths displayed to at most the given number of outputs.

-traverse_disabled

Causes the analysis to include paths that are disabled due to constraints such as cases, **set_disable_timing**, and **set_max_delay**.

-unconstrained

Causes the analysis to include paths that are not normally legal timing paths. These are paths the are -from a pin that does not launch timing paths, or paths that are -to a pin that is not constrained. This option is helpful in debugging situations where the **all_fanin** or **all_fanout** commands return too much information.

-nosplit

Prohibits the breaking of lines in the reports when columns overflow.

-text_only

Prevents the schematic generation of the report requested. With this option, only the text report is generated on the ptc_shell. This option has an effect only when the GUI is in session. Without the GUI, the command always generates only a text report.

-max_paths

Specifies the maximum path count to be reported. The **analyze_paths** command reports the exact number of paths if the number of paths is less than or equal to value of max_paths. Otherwise, it reports the number of paths as greater than value of max_paths. The default value of max_paths is 1000.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **analyze_paths** command shows the number of paths satisfying the given specification, which paths satisfy exceptions, and the clock interactions. The **-traverse_disabled** option can be used to answer the question "why" are there no paths to this endpoint. The **-traverse_disabled** option includes paths that are normally disabled because of set_disable_timing, case settings, constraints that forced internal start pins such as set_max_delay, or loop breaking.

The path counts are split into minimum rise, minimum fall, maximum rise, and maximum fall. The (r,f,R,F) label in the header reminds that the order of counts are: minimum rise, minimum fall, maximum rise, and maximum fall. The small r and f are for the minimum or hold paths. The capital R and F are for the maximum or setup paths.

The constraints column in the report specify the exceptions and other constraints that are satisfied by the paths in that row. The constraints are given identifiers and then printed as footnotes at the end of the report. If there is more than one launching or capture clock, the set of clocks are given an identifier of the form (set #). Then the sets of clocks are listed as footnotes.

The -path_type full option prints the path summary followed by a listing of each pin that is included in any of the paths. The report includes a level number for the pin, and any of the satisfied constraints on a given pin are noted. If the GUI is in session the -path_type full option results in a schematic being generated which would contain the paths reported in the text output.

This command allows for Ctrl-C interruption. If a Ctrl-C interrupt is detected while this command is operating it returns control to the tool shell.

EXAMPLES

The following shows a single endpoint with multiple clock relationships.

```
ptc_shell> analyze_paths -to Tranx/pay_count__reg[1]/D
Path Endpoints:
                           Dominant   Overridden     Count       Clocks
Endpoint                  Constraint Constraints (r,f,R,F)    Launch/Capture
-------------------------------------------------------------------------------
Tranx/pay_count__reg[1]/D                        (5,5,5,5)    (set 0)/(set 0)
Tranx/pay_count__reg[1]/D                        (4,4,4,4)    clk2/(set 0)

Sets of Launching and Capturing Clocks:
  set 0 (2 clocks):
    falling clk
    falling clk2
```

The following example shows a summary report with the following two constraints:

```
  set_multicycle_path 3  -setup -rise -through [get_pins {t/A}]
  set_multicycle_path 1  -hold -rise -through [get_pins {t/Z}]
```

Both constraints only effect the rising paths. The first constraint is dominant for the max (or setup) paths the second constraint is dominant for the min (or hold) rising paths and override the implied -hold constraint of the first one. The falling min and max paths have no constraints.

```
ptc_shell> analyze_paths -through t/Z -path_type summary -from T
```

```
****************************************
Report : analyze_paths
Design : counter
Version: C-2009.06-SP2-Beta1
Date   : Tue Aug 25 20:43:06 2009
****************************************
Summary of Paths:
                                      Dominant   Overridden     Count       Clocks
Startpoint         Endpoint          Constraint Constraints (r,f,R,F)    Launch/Capture
-------------------------------------------------------------------------------
T (in)             ffa/D (FD2)         M0                   (0,0,1,0)    CLK1/CLK
T (in)             ffa/D (FD2)         M1        M0         (1,0,0,0)    CLK1/CLK
T (in)             ffa/D (FD2)                              (0,1,0,1)    CLK1/CLK


Exception Information:
ID    Type             Setup    Hold      Source
-------------------------------------------------------------------------------
M0    multicycle_path  R=3,F=-- --        constraints.tcl, line 13
M1    multicycle_path  --       r=1,f=--  constraint2.tcl, line 44
```

The following example shows paths that were disable because of a set_disable_timing.

```
  ptc_shell> analyze_paths -to TUYMBISTFAILNR -path_type summary -traverse_disabled
Breaking loops for scenario: default
Summary of Paths:
                                      Dominant   Overridden     Count       Clocks
Startpoint         Endpoint          Constraint Constraints (r,f,R,F)    Launch/Capture
-------------------------------------------------------------------------------
GLDEBUGZNR         TUYMBISTFAILNR     F455                 (2,2,2,2)      CLK/CLK
top/m_car_i0/CLK   TUYMBISTFAILNR                          (1,1,1,1)      CLK/CLK
top/m_c08_i0/CLK   TUYMBISTFAILNR                          (1,1,1,1)      CLK/CLK
top/m_c09_i0/CLK   TUYMBISTFAILNR     M503                 (1,1,1,1)      CLK/CLK
top/m_c23_i0/CLK   TUYMBISTFAILNR                          (1,1,1,1)      CLK/CLK
top/m_c45_i0/CLK   TUYMBISTFAILNR                          (1,1,1,1)      CLK/CLK
top/m_c56_i0/CLK   TUYMBISTFAILNR     DT#0                 (1,1,1,1)      CLK/CLK
top/m_c60_i0/CLK   TUYMBISTFAILNR     DT#0                 (1,1,1,1)      CLK/CLK


Exception Information:
ID    Type             Setup Hold  Source
-------------------------------------------------------------------------------
F455  false_path       FALSE FALSE /u/user/scripts/exceptions.sdc, line 1790
M503  multicycle_path  2     0     /u/user/scripts/exceptions.sdc, line 2303

Disabled Object Information:
   DT#0 = set_disable_timing
      at pin: mbist_cntl_i0/U4031/Y
```

The following is an example of a full format type.

```
ptc_shell> analyze_paths -from ffa/CP -to CO -path_type full
****************************************
Report : analyze_paths
Design : counter
Version: C-2009.06-SP2-Beta1
Date   : Tue Aug 25 20:23:11 2009
****************************************
Summary of Paths:
                                      Dominant   Overridden     Count       Clocks
Startpoint         Endpoint          Constraint Constraints (r,f,R,F)    Launch/Capture
-------------------------------------------------------------------------------
ffa/CP (FD2)       CO (out)            F1                   (1,0,1,0)    CLK/VCLK
ffa/CP (FD2)       CO (out)                                 (0,1,0,1)    CLK/VCLK
```

```
Exception Information:
ID    Type           Setup Hold      Source
--------------------------------------------------------------------------------
F1    false_path     r_FALSE r_FALSE  constraints.tcl, line 25

Full report of all pins in the paths

Level Pin                                                    Attr      Constraints
--------------------------------------------------------------------------------
1     ffa/CP (FD2)                                           Start
2     ffa/QN (FD2)
3     m/D (AN4)
4     m/Z (AN4)
5     CO/A (AN2)
6     CO/Z (AN2)
7     CO (out)                                               End       F1
```

SEE ALSO

```
set_false_path(2)
set_multicycle_path(2)
```

analyze_unclocked_pins

Performs analysis of the unclocked register clock pins in the design.

SYNTAX

```
string analyze_unclocked_pins
   [-include include_list]
   [-all_scenarios]
   [-verbose]
   [-nosplit]
   [pin_list]
```

Data Types

```
   include_list      list
   pin_list          list
```

ARGUMENTS

-include *include_list*

Specifies the content to include in the output. The include list might contain any of the following arguments: *summary*, *blocked_clock*, *missing_clock*, *case_disabled*, *register_disabled*, *unclocked*. If the **-include** option is not provided, the default is {summary *blocked_clock missing_clock*}.

-all_scenarios

Considers all of the available scenarios to report unclocked pins. When this option is used, if the clock pin is clocked in at least one of the scenarios, it is not considered as unclocked.

-verbose

When used with the *missing_clock* or *blocked_clock* arguments of the **-include** option, the *-verbose* option reports the list of pins that are unclocked because of the missing or block clock respectively. With the other include options it has no effect.

-nosplit

Prohibits the breaking of lines in the reports when columns overflow.

pin_list

Specifies a list of pin names or a pin collection of unclocked pins to analyze. If no pin_list is given, all unclocked register clock pins in the design are analyzed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Performs analysis of the unclocked register clock pins in the design and prints how many clock pins have the clock reaching them, how many pins are unclocked, how many clock pins are disabled due to case analysis and how many registers are otherwise disabled.

When the *-all_scenarios* option is used, then the report is modified to determine if the clock pin is clocked in at least one scenario and the unclocked pins are pins that are not clocked in any scenario.

EXAMPLES

```
ptc_shell> analyze_unclocked_pin -include {summary}

                Summary
-------------------------------------------
Register Clock Pin Status         Number of Pins
-------------------------------------------
Clocked                           1338934
Unclocked                             397
Case disabled clock pin             31122
Register behavior disabled           2941
-------------------------------------------
Total register clock pins         1373394
```

The "case disabled clock pin" (**case_disabled**) and "register behavior disabled" (**register_disabled**) categories are both registers that have been disabled. For a register to be disabled, all timing checks - setup, hold, recovery, and removal, must be disabled as well as any sequential arcs from clock to Q must also be disabled. If they are disabled because of case settings, they are placed in the "Case disabled clock pin" category. If they are disabled for any other reason such as set_disable_timing, they are placed in the "Register behavior disabled" category. All other unclocked pins are placed in the **unclocked** category.

The **missing_clock** category reports the set of startpoints that fanout to unclocked register clock pins. The start points are printed in the order of the number of unclocked pins they fanout to. These locations might have a missing clock or generated clock definition. If the **missing_clock** option is used with the **verbose** option, a list of unclocked pins that each startpoint fans out to is printed.

The **blocked_clock** category reports if any clocks would reach any of the unclocked pins, but the clocks were blocked by case, a set_disable_timing, a constraint that forced a startpoint, or the command, **set_clock_sense -stop_propagation**. If the **blocked_clock** option is used with the **verbose** option, then a list of the unclocked pins in the fanout of each blocking location is printed.

```
ptc_shell> analyze_unclocked_pin -include {summary} -all_scenarios
```

```
                Summary
-----------------------------------------------------------
Register Clock Pin Status                    Number of Pins
-----------------------------------------------------------
Clocked (in one or more scenarios)                 1339000
Unclocked (in all scenarios)                           322
Case disabled clock pin (in all scenarios)           31100
Register behavior disabled (in all scenarios)         2911
-----------------------------------------------------------
Total register clock pins                          1373333
```

append_to_collection

Adds objects to a collection. Modifies variable.

SYNTAX

```
collection append_to_collection
   var_name
   object_spec
   [-unique]
```

Data Types

```
   var_name         collection
   object_spec      list
```

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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 provides the same semantics as the **add_to_collection** command, 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. See the **add_to_collection** man page for more information about those restrictions.

EXAMPLES

The following example from constraint consistency shows how a collection can be built up with the **append_to_collection** command:

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

apropos

Searches the command database for a pattern.

SYNTAX

```
string apropos
   [-symbols_only]
   pattern
```

Data Types

```
   pattern      string
```

ARGUMENTS

-symbols_only

Searches only command and option names.

pattern

Searches for the specified *pattern*.

DESCRIPTION

The **apropos** command searches the command and option database for all commands that contain the specified *pattern*. The *pattern* argument can include the wildcard characters asterisk (*) and question mark (?). The search is case-sensitive. For each command that matches the search criteria, the command help is printed as though **help -verbose** was used with the command.

Whereas **help** looks only at command names, **apropos** looks at command names, the command one-line description, option names, and option value-help strings. The search can be restricted to only command and option names with the **-symbols_only** option.

When searching for dash options, do not include the leading dash. Search only for the name.

EXAMPLES

In the following example, assume that the **get_cells** and **get_designs** commands have the **-exact** option. Note that without the **-symbols_only** option, the first search picks up commands which have the string "exact" in the one-line description.

```
prompt> apropos exact
 get_cells            # Create a collection of cells
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match cell names against patterns)

 get_designs          # Create a collection of designs
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match design names against patterns)

 real_time            # Return the exact time of day

prompt> apropos exact -symbols_only
 get_cells            # Create a collection of cells
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match cell names against patterns)

 get_designs          # Create a collection of designs
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match design names against patterns)
```

SEE ALSO

help(2)

cell_of

Creates a collection of cells of the given pins. The **cell_of** command is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
string cell_of
   object_list
```

Data Types

```
   object_list      list
```

ARGUMENTS

object_list

Creates a list of pins for which cells to get.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **cell_of** command creates a collection of cells owned by a list of pins. The supported method for getting pins of cells is to use the **get_cells** command with the **-of_objects** option.

SEE ALSO

```
get_cells(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 clocks. 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 an empty 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 constraint consistency you save a collection of ports:

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

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

```
ptc_shell> unset ports
ptc_shell> 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 deleted. For example, if our collection of ports is owned by a design, it is implicitly deleted when the design that owns the ports is deleted. 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 constraint consistency example:

```
ptc_shell> current_design
{"TOP"}

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

ptc_shell> remove_design TOP
Removing design 'TOP'...

ptc_shell> 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.

```
ptc_shell> \
foreach_in_collection s1 $collection {
  echo [get_object_name $s1]
  }
```

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 constraint consistency:

  ```
  ptc_shell> set dports \
  [remove_from_collection [all_inputs] 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 from constraint consistency filters out all leaf cells:

```
ptc_shell> 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 constraint consistency command sorts the ports by direction and then by full name.

```
ptc_shell> 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 constraint consistency:

```
ptc_shell> set_input_delay 3.0 [get_ports in*]
1
ptc_shell> get_ports in*
{"in0", "in1", "in2"}
ptc_shell> query_objects -verbose [get_ports in*]
{"port:in0", "port:in1", "port:in2"}
ptc_shell> 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 **set_input_delay** command. This collection is not the result of the primary command (**set_input_delay**), 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 Gconstraint consistency **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
```

Constraint Consistency Commands

```
all_clocks
all_connected
all_exceptions
all_fanin
all_fanout
all_inputs
all_outputs
all_registers
all_scenarios
get_cells
get_clocks
get_designs
get_exceptions
get_generated_clocks
get_input_delays
get_lib_cells
get_lib_pins
get_libs
get_nets
get_output_delays
get_path_groups
get_pins
get_ports
get_scenarios

Constraint Consistency Variables:

collection_deletion_effort
collection_result_display_limit
```

SEE ALSO

add_to_collection(2)
all_clocks(2)
all_connected(2)
all_exceptions(2)
all_inputs(2)
all_instances(2)
all_outputs(2)
all_scenarios(2)
compare_collections(2)
copy_collection(2)
filter_collection(2)
foreach_in_collection(2)
get_cells(2)
get_clocks(2)
get_designs(2)
get_generated_clocks(2)
get_input_delays(2)
get_output_delays(2)
get_lib_cells(2)
get_lib_pins(2)
get_libs(2)

get_nets(2)
get_path_groups(2)
get_path_pins(2)
get_pins(2)
get_ports(2)
get_scenarios(2)
get_timing_paths(2)
index_collection(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
sort_collection(2)
collection_deletion_effort(3)
collection_result_display_limit(3)

compare_block_to_top

Compares block-level constraints to top-level constraints and identifies inconsistencies.

SYNTAX

```
string compare_block_to_top
   -block_design design
   [-cells cell_list]
   [-top_scenarios top_scenario_list]
   [-block_scenarios block_scenario_list]
   [-include include_list]
   [-use_clock_map]
   [-verbose]
   [-ignore_constant_net_mismatches]
```

Data Types

```
   cell_list           list
   top_scenario_list   list
   block_scenario_list list
   include_list        list
```

ARGUMENTS

-block_design

The block compared with top design.

-cells *cell_list*

List of cell instances to consider. Each of these cells should be instances of the specified block *design*.

-top_scenarios *top_scenario_list*

List of top scenarios to compare. If not specified, the current scenario in the top design is used. If specified, this list must be the same length as the *blockScenarioList*.

-block_scenarios *block_scenario_list*

List of block scenarios to compare. If not specified, the current scenario in the block design is used. If specified, this list must be the same length as the *topScenarioList*.

-include *include_list*

List of content to include in the comparison. By default, all supported constraints are checked. Use this option to specify a subset of constraint types to check. Valid values in this list include boundary, exceptions, clocks, case and misc.

-use_clock_map

Uses custom clock mapping specified by the **set_clock_map** command instead of using clock waveform comparison method.

-verbose

If specified, prints verbose status information.

-ignore_constant_net_mismatches

If specified, ignores violations of case values due to constants propagated from tied nets.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Finds inconsistencies between block-level and top-level constraints. Design teams use various strategies such as budgeting or context characterization to create block-level constraints from top-level constraints. Also, sometimes block-level constraints are propagated to top-level in a bottom-up design flow. This command is a way to check whether the constraints match after such operations. This command produces output in a violation repository (much like the analyze_design output) that can be viewed in the GUI or in the shell by using the *report_constraint_analysis* command.

By default, the behavioral checker used by the comparison engine checks waveforms on clocks reaching the endpoints to determine if same clocks are reaching the endpoint in the top and block scenarios.

This command checks for inconsistencies between top and block-level constraints. To find potential issues within one set of constraints, use the *analyze_design* command at top-level and at block-level.

PrimeTime constraint consistency does not remove subdesigns. You must link your block before running the **compare_block_to_top** command, as shown in the following example.

EXAMPLES

The following example loads the top-level netlist, links the top, applies top-level constraints, then links the block and applies block-level constraints. Finally, it compares constraints for the current scenario of block and top, for all instances of the specified block.

```
ptc_shell> read_verilog ./chip.v
ptc_shell> link_design chip
```

```
ptc_shell> source chip_constraints.tcl
ptc_shell> link_design -add usb_core
ptc_shell> source usb_core_constraints.tcl
ptc_shell> compare_block_to_top -block_design usb_core
```

SEE ALSO

analyze_design(2)
link_design(2)
read_verilog(2)
report_constraint_analysis(2)

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
```

Data Types

```
   collection1   collection
   collection2   collection
```

ARGUMENTS

-order_dependent

Indicates that the order of the objects is to be considered; that is, 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

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **compare_collections** command is used to compare 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 (that is, **compare_collections** considers them identical), and the result is "0".

EXAMPLES

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

```
ptc_shell> compare_collections [get_cells *] [get_cells *]
0
ptc_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
ptc_shell> set c2 [get_cells {u2 u1}]
{"u2", "u1"}
ptc_shell> set c3 [get_cells {u2 u4 u6}]
{"u2", "u4", "u6"}
ptc_shell> compare_collections $c1 $c2
0
ptc_shell> compare_collections $c1 $c2 -order_dependent
-1
ptc_shell> compare_collections $c1 $c3
-1
```

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

```
ptc_shell> set c4 ""
ptc_shell> compare_collections $c1 $c4
-1
ptc_shell> compare_collections $c4 $c4
0
```

SEE ALSO

collections(2)

compare_constraints

Compares two sets of constraints on a single design and identifies inconsistencies.

SYNTAX

```
string compare_constraints
   [-constraints1 scenario_list]
   [-constraints2 scenario_list]
   [-use_clock_map]
```

Data Types

```
   constraints1   list
   constraints2   list
```

ARGUMENTS

-constraints1 ***scenario1_list***

List of scenarios to compare with the list of scenarios in the *constraints2* list. Size of the two lists should be the same. The nth entry in this list is compared with the nth entry in the second list. The scenarios in this list should be on the current_design.

-constraints2 ***scenario2_list***

List of scenarios to compare with the list of scenarios in the *constraints1* list. Size of the two lists should be the same. The nth entry in this list is compared with the nth entry in the first list. The scenarios in this list should be on the current_design.

-use_clock_map

Uses custom clock mapping specified by **set_clock_map** command instead of using the clock waveform comparison method.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **compare_constraints** command finds inconsistencies between two sets of constraints on a design. Manual editing, using tools like PrimeTime to transform and write constraints, changes the constraints. This command checks whether the constraints match after such operations. At the end of the command the GUI populates violations in a violation browser. To get a textual output of the violations, use the **report_constraint_analysis** command.

The comparison engine first compares clock sources to determine if the same clocks are reaching the endpoints. For clocks that are not matched in the first step, a behavioral checker is called by the comparison engine to check waveforms on clocks reaching the endpoints, to determine if same clocks are reaching the endpoint in the two scenarios. For example, if a clock is defined on the input of a buffer in one scenario and on the output of a buffer in another scenario with same waveform, the comparison engine does not issue any violations because the waveforms reaching the endpoints are same.

This command checks for inconsistencies between two sets of constraints. To find potential issues within one set of constraints, use the **analyze_design** command.

EXAMPLES

The following example loads and links the design and applies each set of constraints as a scenario. It then compares the scenarios using the **compare_constraints** command.

```
ptc_shell> read_verilog ./chip.v
ptc_shell> link_design chip
ptc_shell> create_scenario ref
ptc_shell> source chip_ref.tcl
ptc_shell> create_scenario imp
ptc_shell> source chip_imp.tcl
ptc_shell> compare_constraints \
      -constraints1 {ref} -constraints2 {imp}
```

This example shows how a collection of scenarios is compared. In the example, constraints in the **func_ref** scenario are compared with those in **func_impl** and constraints in the **test_ref** scenario are compared with those in **test_impl**.

```
ptc_shell> compare_constraints \
      -constraints1 {func_ref test_ref} \
      -constraints2 {func_impl test_impl}
```

SEE ALSO

```
create_scenario(2)
analyze_design(2)
link_design(2)
read_verilog(2)
source(2)
```

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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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:

```
ptc_shell> set c1 [get_cells "U1*"]
{"U1", "U10", "U11", "U12"}
ptc_shell> 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:

```
ptc_shell> set c1 [get_cells "block1"]
{"block1"}
ptc_shell> query_objects $c2
{"U1", "U10", "U11", "U12"}
```

There might 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.

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

SEE ALSO

collections(2)

cputime

Retrieves the overall user time associated with the current ptc_shell process.

SYNTAX

```
float cputime
   [format]
```

Data Types

```
   format      string
```

ARGUMENTS

format

This argument takes a *printf* style formatting string for a floating point number.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The *cputime* command returns the accumulated user time, in seconds, for the current ptc_shell process. If the *format* string is omitted, an integer number of seconds is returned. If the format string is specified, a floating point number is returned. The number includes fractions of a second elapsed and is formatted in accordance with the given specifier. Refer to the Unix man page for **printf** for a detailed description of how to format a floating point argument.

EXAMPLES

```
ptc_shell> cputime "%g"
3.74
```

SEE ALSO

mem(2)

create_clock

Creates a clock object.

SYNTAX

```
string create_clock
   -period period_value
   [-name clock_name]
   [-waveform edge_list]
   [-add]
   [-comment comment_string]
   [source_objects]
```

Data Types

period_value float *clock_name* string *edge_list* list *source_objects* list *comment_string* string

ARGUMENTS

-period *period_value*

Specifies the clock period in library time units. This is the minimum time over which the clock waveform repeats. The *period_value* must be greater than or equal to zero.

-name *clock_name*

Specifies the name of the clock being created, enclosed in quotation marks. If you do not use this option, the clock gets the same name as the first clock source specified in the *sources* option. If you did not specify *sources*, you must use the **-name** option, which creates a virtual clock not associated with a port or pin. You can use both the **-name** option and the *sources* option to give the clock a more descriptive name than the first source pin or port. If you specify the *-add* option, you must use the **-name** option and the clocks with the same source must have different names.

-waveform *edge_list*

Specifies the rise and fall edge times of the clock waveforms of the clock, in library time units, over an entire clock period. The first time in the *edge_list* is a rising transition, typically the first rising transition after time zero. There must be an even number of edges, and they are assumed to be alternating rise and fall. The edges must be monotonically increasing, except for a special case with two edges, where fall can be less than rise. The numbers should represent one full clock period. If you do not specify this

option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of *period_value*/2.

-add

Specifies whether to add this clock to the existing clock or to overwrite. Use this option to capture the case where multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you specify this option, you must also use the *-name* option.

Defining multiple clocks on the same source pin or port causes longer runtime and higher memory usage than a single clock, because constraint consistency must explore all possible combinations of launch and capture clocks. Use the **set_false_path** command to disable unwanted clock combinations.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

source_objects

Specifies the objects used as sources of the clock. The sources can be ports, pins or nets in the design. If you do not use this option, you must use the **-name** option, which creates a virtual clock not associated with a port, pin or net. If you specify a clock on a pin that already has a clock, the new clock replaces the old one unless you use the **-add** option. When a net is used as the source, the first driver pin of the net is the actual source used in creating the clock.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a clock object. It is created in the current design and is applied to the specified *sources* value. If you do not specify a *sources* value, but give a *clock_name* value, a virtual clock is created. You can create a virtual clock to represent an off-chip clock for input or output delay specification. For more information about input and output delay, see the **set_input_delay** and **set_output_delay** man pages.

If one of the *sources* is already the source of a clock, the source is removed from that clock. This clock is eliminated if it has just one source.

The **create_clock** command also defines the waveform for the clock. The clock can have multiple pulses per period.

The **create_clock** command is used together with the **set_clock_latency**, **set_clock_uncertainty**, **set_propagated_clock**, and **set_clock_transition** commands to specify properties of clock networks. By default, a new path group is created for the clock. This new path group brings together the endpoints related to this clock for cost function calculation. To remove the clock from its assigned group, use the **group_path** command to reassign the clock to another group or to the default path group. For more information, see the **group_path** man page.

The new clock has ideal clock latency and transition time; no propagated delay through the clock network is assumed and a transition time of zero is used at the clock source pin. To enable propagated latency for

a clock network, use the **set_propagated_clock** command. To set an estimated latency, use the **set_clock_latency** command.

To show information about clocks in the design, use the **report_clock** command. To create a collection of clocks matching a pattern and optionally matching filter criteria, use the **get_clocks** command.

To undo **create_clock**, use the **remove_clock** command.

EXAMPLES

The following example creates a clock on a port named *PHI1* with a period of *10.0*, a rise at *5.0*, and a fall at *9.5*.

```
ptc_shell> create_clock PHI1 -period 10 -waveform { 5.0 9.5 }
```

The following example shows a clock named *PHI2* with a falling edge at *5* and a rising edge at *10* with a period of *10*. Because the edges for the **-waveform** option should be ordered as first rise, then fall, and should increase in value, the fall edge can be given as *15*. This places the next falling edge after the first rise edge at *10*.

```
ptc_shell create_clock PHI2 -period 10 -waveform { 10 15 }
```

The following example creates a virtual clock *PHI2* with a period of *10.0*, a rise at *0.0*, and a fall at *5.0*.

```
ptc_shell> create_clock -name PHI2 -period 10 -waveform {0.0 5.0}
```

The following example creates a clock named *clk2* with multiple sources.

```
ptc_shell> create_clock -name clk2 -period 10 \
-waveform {2.0 4.0} {clkgen1/Z clkgen2/Z clkgen3/Z}
```

The following example creates a clock named *CLK* on pin *u13/Z* with a period of *25*, a fall at *0.0*, a rise at *5.0*, a fall at *10.0*, a rise at *15.0*, and so on.

```
ptc_shell> create_clock u13/Z -name CLK -period 25 -waveform { 5 10 15 25}
```

SEE ALSO

```
all_clocks(2)
get_clocks(2)
group_path(2)
remove_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)
set_input_delay(2)
set_output_delay(2)
```

create_command_group

Creates a new command group.

SYNTAX

```
string create_command_group [-info info_text]
group_name
```

ARGUMENTS

-info *info_text*

Help string for the group

group_name

Specifies the name of the new group.

DESCRIPTION

The **create_command_group** command is used to create a new command group, which you can use to separate related user-defined procedures into functional units for the online help facility. When a procedure is created, it is placed in the "Procedures" command group. With the **define_proc_attributes** command, you can move the procedure into the group you created.

The *group_name* can contain any characters, including spaces, as long as it is appropriately quoted. If *group_name* already exists, **create_command_group** quietly ignores the command. The result of **create_command_group** is always an empty string.

EXAMPLES

The following example demonstrates the use of the **create_command_group** command:

```
prompt> create_command_group {My Procedures} -info "Useful utilities"

prompt> proc plus {a b} { return [expr $a + $b] }

prompt> define_proc_attributes plus -command_group "My Procedures"

prompt> help
My Procedures:
  plus

 ...
```

SEE ALSO

define_proc_attributes(2)
help(2)
proc(2)

create_generated_clock

Creates a generated clock object.

SYNTAX

```
string create_generated_clock
   [-name clock_name]
   -source master_pin
   [-divide_by divide_factor | -multiply_by multiply_factor |
    -edges edge_list ]
   [-combinational]
   [-duty_cycle percent]
   [-invert]
   [-preinvert]
   [-edge_shift edge_shift_list]
   [-add]
   [-comment comment_string]
   [-master_clock clock]
   [-pll_output output_pin]
   [-pll_feedback feedback_pin]
   source_objects
```

Data Types

```
   clock_name          string
   master_pin          list
   divide_factor       int
   multiply_factor     int
   percent             float
   edge_list           list
   edge_shift_list     list
   clock               string
   comment_string      string
   source_objects      list
```

ARGUMENTS

-name *clock_name*

Specifies the name of the generated clock. If you do not use this option, the clock receives the same name as the first clock source specified in the **-source** option. If you specify the *-add* option, you must use the **-name** option and the clocks with the same source must have different names.

-source *master_pin*

Specifies the master clock source (a clock source pin in the design) from which the clock waveform is to be derived. Note that the actual delays (latency) for the generated clock are computed using its own source pins and not the *master_pin*.

-divide_by *divide_factor*

Specifies the frequency division factor. If the *divide_factor* value is *2*, the generated clock period is twice as long as the master clock period.

-multiply_by *multiply_factor*

Specifies the frequency multiplication factor. If the *multiply_factor* value is *3*, the generated clock period is one-third as long as the master clock period.

-edges *edge_list*

Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must be not less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, *1* represents the first source edge, *2* represents the second source edge, and so on.

-combinational

The source latency paths for this type of generated clock only includes the logic where the master clock propagates. The source latency paths will not flow through sequential element clock pins, transparent latch data pins or the source pins of other generated clocks.

-duty_cycle *percent*

Specifies the duty cycle, in percentage, if frequency multiplication is used. Duty cycle is the high pulse width.

-invert

Inverts the generated clock signal (in the case of frequency multiplication and division). This option first creates the generated clock and then inverts the generated clock signal.

-preinvert

Creates a generated clock based on the inverted clock signal. This option first inverts the signal and then creates the generated clock signal.

-edge_shift *edge_shift_list*

Specifies a list of floating point numbers that represents the amount of shift, in library time units, that the specified edges are to undergo to yield the final generated clock waveform. The number of edge_shifts specified must be equal to the number of edges specified. The values can be positive or negative, positive indicating a shift later in time, negative a shift earlier. For example, *1* indicates that the corresponding edge is to be shifted by one library time unit.

-add

Specifies whether to add this clock to the existing clock or to overwrite. Use this option to capture the case where multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the

master pin. If you specify this option, you must also use the **-name** and **-master_clock** options.

Defining multiple clocks on the same source pin or port causes longer runtime and higher memory usage than a single clock, because constraint consistency must explore all possible combinations of launch and capture clocks. Use the **set_false_path** command to disable unwanted clock combinations.

-master_clock ***clock***

Specifies the master clock to be used for this generated clock if multiple clocks fan into the master pin. If you specify this option, you must also use the **-add** option.

-pll_output ***output_pin***

Specifies the output pin of the PLL which is connected to the feedback pin. For single output PLLs, this pin is same as the pin on which the generated clock is defined.

-pll_feedback ***feedback_pin***

Specifies the feedback pin of the PLL. There should be a path in the circuit connecting one of the outputs of the PLL to this feedback pin.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

source_objects

Specifies a list of ports, pins or nets defined as generated clock source objects. When a net is used as the source, the first driver pin of the net is the actual source used in creating the generated clock.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a generated clock object. It is created in the current design. The command creates it in the current design and defines a list of objects as generated clock sources in the current design. You can specify a pin or a port as a generated clock object. The command also specifies the clock source from which it is generated. The advantage of using this command is that whenever the master clock changes, the generated clock automatically changes.

The generated clock can be created as a frequency divided clock (by using the **-divide_by** option), frequency multiplied clock (by using the **-multiply_by** option), special divide by one (by using the **-combinational** option), or an edge-derived clock (by using the **-edges** option). The frequency-divided or frequency-multiplied clock can be inverted by using the **-invert** option. The shifting of edges of the edge-derived clock is specified by using the **-edge_shift** option. The **-edge_shift** option is used for intentional edge shifts and not for clock latency.

The number of edges specified by **-edges** to make one period of the generated clock waveform must be an odd number equal to or greater than 3. For example, in the following command:

create_generated_clock -source clk **-edges** { 1 3 5 } [get_pins flop/Q]

edge 1 indicates the first rising of the generated clock, edge 3 indicates the first falling of the generated clock, and edge 5 indicates the next rising of the generated clock. Note that the period of the generated clock is determined by the final entry in the edge list.

Non-increasing edges as -edges { 1 1 3 } is allowed and usually used with -edge_shift to produce a generated clock pulse independent of the duty cycle of the master clock itself.

If a generated clock is specified with a *divide_factor* value that is a power of 2 (1, 2, 4, ...), the rising edges of the master clock are used to determine the edges of the generated clock. If the *divide_factor* value is not a power of two, the edges are scaled from the master clock edges.

Using the **create_generated_clock** command on an existing **generated_clock** object overwrites its attributes. The **generated_clock** objects are expanded to real clocks at the time of analysis.

The following commands can reference the generated_clock: **set_clock_latency, set_clock_uncertainty, set_propagated_clock**, and **set_clock_transition**.

For internally generated clocks, constraint consistency automatically computes the clock source latency if the master clock of the generated clock has propagated latency and no user-specified value for generated clock source latency exists. If the master clock is ideal and has source latency, and there is no user-specified value for the generated clock's source latency, then zero source latency is assumed.

To display information about generated clocks, use the **report_clock** command.

EXAMPLES

The following example creates a frequency divide_by 2 generated clock.

```
ptc_shell> create_generated_clock -divide_by 2 \
-source [get_pins CLK] [get_pins mydesign]
```

The following example creates a frequency divide_by *3* generated_clock. If the master clock period is *30*, and master waveform is *{24 36}*, the generated clock period is *90* with waveform *{72 108}*.

```
ptc_shell> create_generated_clock -divide_by 3 \
-source [get_pins CLK] [get_pins div3/Q]
```

The following example creates a frequency multiply_by *2* generated_clock with a duty cycle of *60*%.

```
ptc_shell> create_generated_clock -multiply_by 2 -duty_cycle 60 \
-source [get_pins CLK] [get_pins mydesign1]
```

The following example creates a frequency multiply_by *3* generated_clock with a duty cycle equal to the master clock duty cycle. If the master clock period is *30*, and master waveform is *{24 36}*, the generated clock period will be *10* with waveform *{8 12}*.

```
ptc_shell> create_generated_clock -multiply_by 3 \
-source [get_pins CLK] [get_pins div3/Q]
```

The following example creates a generated clock whose edges are edges *1*, *3*, and *5* of the master clock source.

```
ptc_shell> create_generated_clock -edges {1 3 5} \
-source [get_pins CLK] [get_pins mydesign2]
```

The following example shows the generated clock in the previous example with each derived edge shifted by *1* time unit.

```
ptc_shell> create_generated_clock -edges {1 3 5} -edge_shift {1 1 1} \
-source [get_pins CLK] [get_pins mydesign2]
```

The following example creates an inverted clock.

```
ptc_shell> create_generated_clock -divide_by 1 -invert
```

The following example creates a rising edge pulse triggered by the rising edge of its master clock.

```
ptc_shell> create_generated_clock -edges {1 1 3} -edge_shift {0 5 0} \
-source [get_pins CLK] [get_pins mydesign2]
```

The following example creates a falling edge pulse triggered by the rising edge of its master clock with a period 10.

```
ptc_shell> create_generated_clock -edges {1 1 3} -edge_shift {0 5 0} \
-invert -source [get_pins CLK] [get_pins mydesign2]
```

The following example creates a frequency doubling pulse. A rising edge pulse triggered by both rising and falling edges of its master clock with a period of 10.

```
ptc_shell> create_generated_clock -edges {1 1 2 2 3} -edge_shift {0 2.5 0 2.5 0} \
-source [get_pins CLK] [get_pins mydesign2]
```

SEE ALSO

check_timing(2)
create_clock(2)
get_generated_clocks(2)
remove_generated_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
set_propagated_clock(2)

create_operating_conditions

Creates a new set of operating conditions in a library.

SYNTAX

```
int create_operating_conditions
   -name name -library library_name
   -process process_value -temperature temperature_value
   -voltage voltage_value [-tree_type tree_type]
   [-calc_mode calc_mode]
   [-rail_voltages rail_value_pairs]
```

Data Types

```
   name                 string
   library_name         string
   process_value        float
   temperature_value    float
   voltage_value        float
   tree_type            string
   calc_mode            string
   rail_value_pairs     Tcl list
```

ARGUMENTS

"operating conditions, creating"

-name *name*

Specifies the name of the new set of operating conditions.

-library *library_name*

Specifies the name of the library for the new operating conditions.

-process *process_value*

Specifies the process scaling factor for the operating conditions. Allowed values are 0.0 through 100.0.

-temperature *temperature_value*

Specifies the temperature value, in degrees Celsius, for the operating conditions. Allowed values are -

300.0 through +500.0.

-voltage ***voltage_value***

Specifies the voltage value, in volts, for the operating conditions. Allowed values are 0.0 through 1000.0.

-tree_type ***tree_type***

Specifies the tree type for the operating conditions. Allowed values are *best_case_tree*, *balanced_tree* (the default), or *worst_case_tree*. The tree type is used to estimate interconnect delays by providing a model of the RC tree.

-calc_mode ***calc_mode***

For use only with DPCM libraries. Specifies the DPCM delay calculator mode for the operating conditions; analogous to the *process* used in Synopsys libraries. Allowed values are *unknown* (the default), *best_case*, *nominal*, or *worst_case*. The default behavior (*unknown*) is to use worst case values during analysis similarly to worst_case. If **-rail_voltages** are specified, the command sets all (worst_case, nominal, and best_case) voltage values.

-rail_voltages ***rail_value_pairs***

Specifies a list of name-value pairs that defines the voltage for each specified rail. The name is one of the rail names defined in the library; the value is the voltage to be assigned to that rail. By default, rail voltages are as defined in the library; use this option to override the default voltages for specified rails.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a new set of operating conditions in the specified library. A technology library contains a fixed set of operating conditions; this command allows you to create new, additional operating conditions.

To see the operating conditions defined for a library, use the **report_lib** command.

To set operating conditions on the current design, use **set_operating_conditions**.

EXAMPLES

The following example creates a new set of operating conditions called WC_CUSTOM in the library "tech_lib", specifying new values for process, temperature, voltage, and tree type. By default, rail voltages remain as defined in the library.

```
ptc_shell> create_operating_conditions -name WC_CUSTOM \
-library tech_lib -process 1.2 -temperature 30.0 -voltage 2.8 \
-tree_type worst_case_tree
```

The following example creates a new set of operating conditions called OC3, in the library "IBM_CMOS5S6_SC", specifying new values for process, temperature, voltage, and rail voltages for rails VTT and VDDQ. By default, the tree type *balanced_tree* is used.

```
ptc_shell> create_operating_conditions -name OC3 \
-lib IBM_CMOS5S6_SC -proc 1.0 -temp 100.0 -volt 4.0 \
-rail_voltages {VTT 3.5 VDDQ 3.5}
```

SEE ALSO

set_operating_conditions(2)
report_lib(2)

create_rule

Creates a user-defined rule. Rules are checked to identify potential problems in constraints or in the design.

SYNTAX

```
Boolean create_rule
   -name name
   [-severity severity]
   -message message_list
   [-parameters parameter_list]
   [-description description]
   [-global]
   -checker_proc user_defined_checker
```

Data Types

name	string
severity	string
message_list	list
parameter_list	list
description	string
user_defined_checker	string

ARGUMENTS

-name *name*

Specifies a unique name for the rule being created.

-severity *severity*

Specifies the severity of a violation of this rule. Valid values are: "fatal", "error", "warning" or "info". If not specified, the severity of the rule is "warning".

-message *message_list*

Specifies the predefined text fragments which, along with the parameter values, make up the message text for a violation. For example, a rule that prints a violation for one clock might have *message_list* such as {"Clock " " is created on a pin instead of a port."}. In this case, one parameter (the clock name) would be defined.

-parameters ***parameter_list***

Specifies the names of parameters which are used in conjunction with the *message_list* to create the message text for a violation. For example, a rule that needs to print the clock name as a parameter might have *parameter_list* of {"clock"}.

-description ***description***

The description is a string enclosed in double quotes which provides more information about the rule. It can describe why a violation is problematic, and suggest ways to address the problem.

-global

Indicates that this rule is intended to be checked in user-defined global checking procedures only. This means that the rule is scenario independent and is checked only one time and not per scenario. Without **-global**, the rule is assumed to be checked in user-defined scenario checking procedure only.

-checker_proc ***user_defined_checker***

Associate the user-defined rule with an existing user-defined checker procedure. The checker procedure is automatically invoked in the analyze_design command if the user-defined rule is enabled and included in the rules checked by the analyze_design command. A user-defined checker procedure can only issue violations of user-defined rules associated with the checker. If the user-defined rule is a scenario rule, the user-defined checkers is invoked during scenario checks. If the user-defined rule is a global rule, the user-defined checker is invoked during global checks. All user-defined rules associated with a user-defined checker procedure must be either scenario rules or global rules. A checker procedure cannot have both scenario rules and global rules associated with it.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a rule for checking in the **analyze_design** command. A rule is used for formatting output related to potential problems in the design or in the constraints for a scenario. You can pass a user-defined rule checking procedure to the **analyze_design** command. In that procedure, you can call the **create_rule_violation** command to register a violation of a rule that you defined using the **create_rule** command.

To remove a user-defined rule, use the **remove_rule** command. To temporarily disable one or more rules, use the **disable_rule** command. You can specify properties for the rule using the **set_rule_property** command.

EXAMPLES

The following example creates a rule related to clocks with two parameters:

```
ptc_shell> create_rule -name CLK_0001 -severity warning \
```

```
        -message {"Clock " " has " " source(s) on inout ports"}  \
        -parameters {"clock" "count"} \
        -description "A clock defined on an inout port may become \
            invalid when the port is used in output mode."
        -checker_proc my_checker
```

SEE ALSO

analyze_design(2)
create_rule_violation(2)
get_rules(2)
disable_rule(2)
enable_rule(2)
remove_rule(2)
set_rule_property(2)

create_rule_violation

Creates a violation entry for the specified rule in user-defined checking procedures.

SYNTAX

```
string create_rule_violation
   -rule rule_name
   -parameter_values value_list
   -details details_text
```

Data Types

```
   rule_name      string
   value_list     list
   detail_text    string
```

ARGUMENTS

-rule *rule_name*

Specifies a unique name for the rule being violated.

-parameter_values *value_list*

Specifies a list of values which are used in conjunction with the *message_list* specified in *create_rule* to create the message text for a violation. For example, a rule that needs to print the clock name as a parameter might have parameter values of {"clock1"}.

-details *detail_text*

The details is a string enclosed in double quotes which provides more information about the rule violation. It can describe why a violation is problematic, and suggest ways to address the problem.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a rule violation entry in user-defined checking procedures. It is only active within the **analyze_design** command. **create_rule_violation** should be used only in user-defined rule checking procedures.

EXAMPLES

The following example creates a rule violation of rule CAP_0001 in a user-defined checking procedure:

```
proc test_user_scenario_rule {} {
  create_rule_violation -rule CAP_0001 -parameter_values {"my_port"} \
    -details "some details here"
}
```

SEE ALSO

create_rule(2)
analyze_design(2)

create_ruleset

Creates a set of related rules

SYNTAX

```
Boolean create_ruleset
   -name name
   rule_list
```

Data Types

```
   name        string
   rule_list   list
```

ARGUMENTS

-name *name*

Specifies a unique name for the ruleset being created.

rule_list

The list of rules or rulesets to include in this ruleset. If a ruleset is specified, all rules from that ruleset are included.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a named set of rules for this session. Rulesets can be passed to other commands which operate on rules.

It is an error to specify the same name as an existing ruleset. If you need to redefine an existing ruleset, first remove the old one with the **remove_ruleset** command and then create a new one containing the wanted rules.

EXAMPLES

The following example creates a ruleset named "user_clock_rules" and uses it in another command.

```
ptc_shell> create_ruleset -name user_clock_rules \
    [get_rules "UDEF_CLK_*"]
ptc_shell> enable_rule user_clock_rules
```

SEE ALSO

analyze_design(2)
create_rule(2)
get_rules(2)
enable_rule(2)
remove_ruleset(2)

create_scenario

Creates a scenario in the current design. The **create_mode** command is a synonym for the **create_scenario** command.

SYNTAX

```
string create_scenario
   scenario_name
   [-corner corner_name]
```

Data Types

```
   scenario_name   string
   corner_name     string
```

ARGUMENTS

scenario_name

Specifies the name of the scenario being created.

-corner *corner_name*

Specifies the corner comprising the scenario. This is currently used only for mode merging.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a scenario in the current design. Scenarios are used to contain a set of constraints and operating environment for the design. Unique scenarios are often used for different combinations of operating modes, process/temperature/voltage conditions, and power schemes.

Constraints are contained in the current scenario. When the design is linked, a single scenario named "default" is automatically created and set to be the current scenario. If the **create_scenario** command is

used, the current scenario is set to the newly-created scenario. You can set the current scenario to a different scenario with the **current_scenario** command.

To create a collection of scenarios matching a pattern and optionally matching filter criteria, use the **get_scenarios** command. To get a collection of all of the scenarios in the design, use the **all_scenarios** command.

To undo the **create_scenario** command, use the **remove_scenario** command.

EXAMPLES

The following example creates a scenario named "Mission5" and sets it to be the current scenario.

```
ptc_shell> create_scenario Mission5
```

SEE ALSO

all_scenarios(2)
current_scenario(2)
get_scenarios(2)
remove_scenario(2)

create_waiver

Creates a violation waiver. Waivers can conditionally suppress violations of an enabled rule.

SYNTAX

```
Boolean create_waiver

    [-name name]
    [-design design]
    [-scenario scenario]
    -rule rule_name
    [-condition condition]
    [-not_condition condition]
    [-cells cell_list]
    [-all_scenarios]
    [-comment comment]
```

Data Types

```
name          string
design        string
scenario      string
rule_name     string
condition     list
cells         list
comment       string
```

ARGUMENTS

-name *name*

Specifies a name for the waiver being created. If there is an existing waiver with this name, it is overwritten. If no name is given, a unique name is automatically generated for the waiver.

-design *design*

Specifies the design to which the waiver is applied. If no design is given, the waiver is created for the current design. Expects linked design to be passed.

-scenario *scenario*

Specifies the scenario to which the waiver is applied. If no scenario is given, the waiver is created for

the current scenario.

-rule *rule_name*

Specifies the rule. The waiver conditionally suppresses violations of the given rule.

-condition *condition*

There can be multiple *-condition* options. Each *-condition* specifies one requirement for the violation to be suppressed. *condition* is a list, where the first element is the name of a rule parameter, and all remaining elements are collections of netlist/constraint objects or strings. A *-condition* option is satisfied if the value of the violation parameter falls in one of the collections or strings. A violation is suppressed only if all the *-condition* and *-not_condition* options are satisfied.

-not_condition *condition*

There can be multiple *-not_condition* options. Each *-not_condition* specifies one requirement for the violation to be suppressed. *condition* is a list, where the first element is the name of a rule parameter, and all remaining elements are collections of netlist/constraint objects or strings. A *-not_condition* option is satisfied if the value of the violation parameter does not fall in any of the collections or strings. A violation is suppressed only if all the *-condition* and *-not_condition* options are satisfied.

-cells *cell_list*

List of cells for instance based waivers. The list can contain hierarchical or leaf cells. For leaf cells, the violations that affect only such instances are waived. For hierarchical cells, the violations that affect only objects inside or on the hierarchical instance are waived.

-all_scenarios

Use this option in conjunction with *-cells* option to indicate that the instance based waiver should apply to all the scenarios and not just the scenario specified by the "-scenario" option.

-comment *comment*

A comment on the waiver (up to 1024 characters).

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a violation waiver for the given rule. A violation waiver suppresses a violation if the violation satisfies all *-condition* and *-not_condition* conditions. Multiple waivers can be created on one rule. A violation is suppressed if it meets the requirements of at least one waiver.

If the rule is scenario-dependent, the waiver is active when checks are performed in the scenario specified by the "-scenario" option, or the current scenario if no scenario is specified; if the rule is scenario-independent, the waiver is active in scenario-independent checks only.

To remove a violation waiver use the **remove_waiver** command.

EXAMPLES

The following example creates a waiver for CLK_0003 violations if the **clock** parameter is CLK1 or CLK2, and if the **pin** parameter is not U1/A or U1/B:

```
ptc_shell> create_waiver -name my_waiver1 -rule CLK_0003 \
    -condition [list "clock" [get_clocks {CLK1 CLK2}]] \
    -not_condition [list "pin" [get_pins U1/A] [get_pins U1/B]] \
    -comment "waiver example"
```

The following example creates a waiver for all objects inside block instance hier_inst. Also, it specifies that the waiver should be applied to all scenarios.

```
ptc_shell> create_waiver -name my_waiver1 -cells [get_cell hier_inst] \
    -all_scenarios \
    -comment "waiver example2"
```

When creating a waivers for block-to-top and SDC-to-SDC rules, *current_design* and *current_scenario* can be used to switch the context under which the object is to be retrieved. The following example shows how to create a waiver for B2T_CLK_0012 rule.

```
ptc_shell> create_waiver -rule B2T_CLK_0012 \
    -condition [list "blk_object" [current_design BLOCK; get_pins zGate/A]] \
    -condition [list "blk_object_type" "pin"] \
    -condition [list "top_object" [current_design TOP; get_pins b1/zGate/A]] \
    -condition [list "top_object_type" "pin"]
```

The following example shows how to create a waiver for the S2S_CLK_0001 rule.

```
ptc_shell> create_waiver -rule S2S_DIS_0001 \
    -condition [list "object" [current_design S2S_CLK_0001; get_pins ck2_i2]] \
    -condition [list "object_type" "pin"] \
    -condition [list "scenario1" "set2"] \
    -condition [list "scenario2" "set1"]
```

SEE ALSO

analyze_design(2)
report_waiver(2)
remove_waiver(2)
write_waiver(2)

current_design

Sets or gets the current design in constraint consistency.

SYNTAX

```
string current_design
   [design_name]
```

Data Types

```
   design_name      string
```

ARGUMENTS

design_name

Specifies the working or focal design for many constraint consistency commands. If *design_name* is not specified, **current_design** returns a collection containing the current design. If *design_name* refers to a design that cannot be found, an error is issued and the working design remains unchanged.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets or gets the working design for many constraint consistency commands. Without arguments, **current_design** returns a collection containing the current working design. The combination of the current design and current instance defines the context for many constraint consistency commands.

To display designs currently available in constraint consistency, use **list_designs**.

EXAMPLES

The following example uses **current_design** to show the current context and to change the context from one design to another.

```
ptc_shell> current_design
{"TOP"}

ptc_shell> list_designs

Design Registry:
    ADDER             /designs/dbs/my_design.db:ADDER
    FULL_ADDER        /designs/dbs/my_design.db:FULL_ADDER
    FULL_SUBTRACTOR   /designs/dbs/my_design.db:FULL_SUBTRACTOR
    HALF_ADDER        /designs/dbs/my_design.db:HALF_ADDER
    HALF_SUBTRACTOR   /designs/dbs/my_design.db:HALF_SUBTRACTOR
    SUBTRACTOR        /designs/dbs/my_design.db:SUBTRACTOR
 *  TOP               /designs/dbs/my_design.db:TOP

ptc_shell> current_design ADDER
{"ADDER"}

ptc_shell> current_design
{"ADDER"}
```

The **current_design** command can be used as a parameter to other **ptc_shell** commands. In the following example, **remove_design** is used to delete the working design from **ptc_shell**.

```
ptc_shell> current_design
{"TOP"}
ptc_shell> remove_design [current_design]
Removing design 'TOP'...
1
ptc_shell> current_design
Error: Current design is not defined. (DES-001)
ptc_shell>
```

SEE ALSO

current_instance(2)
list_designs(2)

current_instance

Sets the working instance object in **ptc_shell** and enables other commands to be used relative to a specific instance in the design hierarchy.

SYNTAX

```
string current_instance
   [instance]
```

Data Types

```
   instance      string
```

ARGUMENTS

instance

Specifies the working instance (cell) in **ptc_shell**. If *instance* is not specified, the focus returns to the top level of the hierarchy in the current design. If *instance* is ".", the current instance remains unchanged. If *instance* is "..", the context is moved up one level in the instance hierarchy. If *instance* begins with "/", **ptc_shell** sets both the current design and the current instance. More complex examples of *instance* arguments are described in EXAMPLES.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **current_instance** command sets the working instance in constraint consistency. An instance is a cell in the hierarchy of a design. This command differs from **current_design**, which changes the working design, then sets the current instance to the top level of the new current design. The combination of the current design and current instance defines the context for many constraint consistency commands.

current_instance traverses the design hierarchy similar to the way the UNIX **"cd"** command traverses the file hierarchy. **current_instance** operates with a variety of *instance* arguments:

- If no *instance* argument is specified, the focus of ptc_shell is returned to the top level of the hierarchy.
- If *instance* is ".", the current instance is returned and no change is made.
- If *instance* is "..", the current instance is moved up one level in the design hierarchy.
- If *instance* is a valid cell (or cell name) at the current level of hierarchy, the current instance is moved down to that level of the design hierarchy.
- Multiple levels of hierarchy can be traversed in a single call to **current_instance** by separating multiple cell names with slashes. For example, **current_instance U1/U2** sets the current instance down two levels of hierarchy.
- The ".." directive can also be nested in complex *instance* arguments. For example the command **current_instance "../../MY_INST"** attempts to move the context up two levels of hierarchy, then down one level to the **"MY_INST"** cell.

EXAMPLES

The following example uses **current_instance** to move up and down the design hierarchy. The **all_instances** command is also used to list instances of a specific design relative to the current instance.

```
ptc_shell> current_design TOP
{"TOP"}

ptc_shell> current_instance U1
U1

ptc_shell> current_instance "."
U1

ptc_shell> query_objects [all_instances ADDER]
{"U1", "U2", "U3", "U4"}

ptc_shell> current_instance U3
U1/U3

ptc_shell> current_instance "../U4"
U1/U4

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

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

```
ptc_shell> current_design
{"TOP"}

ptc_shell> current_instance "U2/U1"
U2/U1

ptc_shell> current_design ADDER
{"ADDER"}
```

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

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.

```
ptc_shell> current_design
{"TOP"}

ptc_shell> current_instance U1
U1

ptc_shell> current_instance "/TOP/U2"
U2

ptc_shell> current_instance "/ALARM_BLOCK/U6"
U6

ptc_shell> current_design
{"ALARM_BLOCK"}
```

SEE ALSO

```
all_instances(2)
current_design(2)
list_designs(2)
query_objects(2)
```

current_scenario

Sets a scenario in the current session to be current. Constraint creation and modification commands apply to the current scenario. The **current_mode** command is a synonym for the **current_scenario** command.

SYNTAX

```
int current_scenario
   [scenario_name]
```

Data Types

```
   scenario_name     string
```

ARGUMENTS

scenario_name

Specifies the name of a defined scenario in the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets one scenario in the current session to be current. Constraint creation and modification commands apply to the current scenario.

EXAMPLES

In the following example, two nondefault scenarios are created, and the **current_scenario** command is

set to "Mission1".

```
1
ptc_shell> create_scenario Mission1
1
ptc_shell> create_scenario Test1
1
ptc_shell> current_scenario Mission1
{"Mission1"}
ptc_shell> source Mission1_constraints.tcl
1
```

SEE ALSO

create_scenario(2)
remove_scenario(2)
source(2)

date

Returns a string containing the current date and time.

SYNTAX

```
string date
```

DESCRIPTION

The **date** command generates a string containing the current date and time, and returns that string as the result of the command. The format is fixed as follows:

```
ddd mmm nn hh:mm:ss yyyy
```

Where:

```
ddd is an abbreviation for the day
mmm is an abbreviation for the month
nn is the day number
hh is the hour number (24 hour system)
mm is the minute number
ss is the second number
yyyy is the year
```

The **date** command is useful because it is native. It does not fork a process. On some operating systems, when the process becomes large, no further processes can be forked from it. With it, there is no need to call the operating system with **exec** to ask for the date and time.

EXAMPLES

The following command prints the date.

```
prompt> echo "Date and time: [date]"
Date and time: Thu Dec  9 17:29:51 1999
```

SEE ALSO

exec(2)

define_proc_attributes

Defines attributes of a Tcl procedure, including an information string for help, a command group, a set of argument descriptions for help, and so on. The command returns the empty string.

SYNTAX

```
string define_proc_attributes
   proc_name
   [-info info_text]
   [-define_args arg_defs]
   [-define_arg_groups group_defs]
   [-command_group group_name]
   [-return type_name]
   [-hide_body]
   [-hidden]
   [-dont_abbrev]
   [-permanent]
```

Data Types

```
   proc_name      string
   info_text      string
   arg_defs       list
   group_defs     list
   group_name     string
   type_name      string
```

ARGUMENTS

proc_name

Specifies the name of the existing procedure.

-info *info_text*

Provides a help string for the procedure. This is printed by the **help** command when you request help for the procedure. If you do not specify *info_text*, the default is "Procedure".

-define_args *arg_defs*

Defines each possible procedure argument for use with **help -verbose**. This is a list of lists where each list element defines one argument.

-define_arg_groups ***group_defs***

Defines argument checking groups. These groups are checked for you in parse_proc_arguments. each list element defines one group

-command_group ***group_name***

Defines the command group for the procedure. By default, procedures are placed in the "Procedures" command group.

-return ***type_name***

Specifies the type of value returned by this proc. Any value may be specified. Some applications use this information for automatically generated dialogs etc. Please see the type_name option attribute described below.

-hide_body

Hides the body of the procedure from **info body**.

-hidden

Hides the procedure from **help** and **info proc**.

-dont_abbrev

Specifies that the procedure can never be abbreviated. By default, procedures can be abbreviated, subject to the value of the **sh_command_abbrev_mode** variable.

-permanent

Defines the procedure as permanent. You cannot modify permanent procedures in any way, so use this option carefully.

-deprecated

Defines the procedure as deprecated i.e. it should no longer be called but is still available.

-obsolete

Defines the procedure as obsolete i.e. it used to exist but can no longer be called.

DESCRIPTION

The **define_proc_attributes** command associates attributes with a Tcl procedure. These attributes are used to define help for the procedure, locate it in a particular command group, and protect it.

When a procedure is created with the **proc** command, it is placed in the Procedures command group. There is no help text for its arguments. You can view the body of the procedure with **info body**, and you can modify the procedure and its attributes. The **define_proc_attributes** command allows you to change these aspects of a procedure.

Note that the arguments to Tcl procedures are all named, positional arguments. They can be programmed with default values, and there can be optional arguments by using the special argument name *args*. The

define_proc_attributes command does not relate the information that you enter for argument definitions with **-define_args** to the actual argument names. If you are describing anything other than positional arguments, it is expected that you are also using **parse_proc_arguments** to validate and extract your arguments.

The *info_text* is displayed when you use the **help** command on the procedure.

Use **-define_args** to define help text and constraints for individual arguments. This makes the help text for the procedure look like the help for an application command. The value for **-define_args** is a list of lists. Each element has the following format:

```
arg_name option_help value_help data_type attributes
```

The elements specify the following information:

- *arg_name* is the name of the argument.
- *option_help* is a short description of the argument.
- *value_help* is the argument name for positional arguments, or a one word description for dash options. It has no meaning for a Boolean option.
- *data_type* is optional and is used for option validation. The *data_type* can be any of: string, list, boolean, int, float, or one_of_string. The default is string.
- *attributes* is optional and is used for option validation. The *attributes* is a list that can have any of the following entries:
    - "required" - This argument must be specified. This attribute is mutually exclusive with optional.
    - "optional" - Specifying this argument is optional. This attribute is mutually exclusive with "required."
    - "value_help" - Indicates that the valid values for a one_of_string argument should be listed whenever argument help is shown.
    - "values {<list of allowable values>}" - If the argument type is one_of_string, you must specify the "values" attribute.
    - "type_name <name>" - Give a descriptive name to the type that this argument supports. Some applications may use this information to provide features for automatically generated dialogs, etc. Please see product documentation for details. This attribute is not supported on boolean options.
    - "merge_duplicates" - When this option appears more than once in a command, its values are concatenated into a list of values. The default behavior is that the right-most value for the option specified is used.
    - "remainder" - Specifies that any additional positional arguments should be returned in this option. This option is only valid for string option types, and by default the option is optional. You can require at least one item to be specified by also including the required option.
    - "deprecated" - Specifying this option is deprecated i.e. it should no longer be used but is still available. A warning will be output if this option is specified. This attribute cannot be combined with obsolete or required.
    - "obsolete" - Specifying this option is obsolete i.e. it used to exist but can no longer be used. A

warning will be output if this option is specified. This attribute cannot be combined with deprecated or required.

- "min_value" <value> - Specify the minimum value for this option. This attribute is only valid for integer and float types.
- "max_value" <value> - Specify the maximum value for this option. This attribute is only valid for integer and float types.
- "default" <value> - Specify the default value for this option. This attribute is only valid for string, integer and float option types. If the user does not specify this option when invoking the command this default value will be automatically passed to the associated tcl procedure.

The default for *attributes* is "required."

Use the **-define_arg_groups** to define argument checking groups. The format of this option is a list where each element in the list defines an option group. Each element has the following format:

{<type> {<opt1> <opt2> ...} [<attributes>]}

The types of groups are

- *"exclusive"* Only one option in an exclusive group is allowed. All the options in the group must have the same required/optional status. This group can contain any number of options.
- *"together"* If the first option in the group is specified then the second argument is also required. This type of group can contain at most two options.
- \fi"related" These options are related to each other. An automatic dialog builder for this command may try to group these options together.

The supported attributes are:

- *{"label" <text>}* An optional label text to identify this group. The label may be used by an application to automatically build a grouping in a generated dialog.
- *"bidirectional"* This is only valid for a together group. It means that both options must be specified together.

Change the command group of the procedure using the **-command_group** command. Protect the contents of the procedure from being viewed by using **-hide_body**. Prevent further modifications to the procedure by using **-permanent**. Prevent abbreviation of the procedure by using **-dont_abbrev**.

EXAMPLES

The following procedure adds two numbers together and returns the sum. For demonstration purposes, unused arguments are defined.

```
prompt> proc plus {a b} { return [expr $a + $b]}

prompt> define_proc_attributes plus -info "Add two numbers" \
? -define_args {
  {a "first addend" a string required}
```

```
  {b "second addend" b string required}
  {"-verbose" "issue a message" "" boolean optional}}

prompt> help -verbose plus
Usage: plus    # Add two numbers
    [-verbose]         (issue a message)
    a                  (first addend)
    b                  (second addend)

prompt> plus 5 6
11
```

In the following example, the argHandler procedure accepts an optional argument of each type supported by **define_proc_attributes**, then displays the options and values received. Note the only one of -Int, -Float, or -Bool may be specified to the command:

```
proc argHandler {args} {
  parse_proc_arguments -args $args results
  foreach argname [array names results] {
    echo $argname = $results($argname)
  }
}

define_proc_attributes argHandler \
  -info "Arguments processor" \
  -define_args {
     {-Oos "oos help"       AnOos   one_of_string
        {required value_help {values {a b}}}}
     {-Int "int help"       AnInt   int     optional}
     {-Float "float help"   AFloat  float   optional}
     {-Bool "bool help"     ""      boolean optional}
     {-String "string help" AString string  optional}
     {-List "list help"     AList   list    optional}
     {-IDup "int dup help"  AIDup   int     {optional merge_duplicates}}
   } \
  -define_arg_groups {
     {exclusive {-Int -Float -Bool}}
    }
```

SEE ALSO

help(2)
info(2)
parse_proc_arguments(2)
proc(2)
sh_command_abbrev_mode(3)

disable_rule

Disables specified built-in or user-defined rules.

SYNTAX

```
Boolean disable_rule
   rule_list
```

Data Types

```
   rule_list     list
```

ARGUMENTS

rule_list

Specifies the list of rules or rule sets to disable. If a rule set is specified, all rules from that rule set are disabled.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Marks specified rules as being disabled. During **analyze_design**, violations are not created for disabled rules.

EXAMPLES

The following example disables rule "CLK_0001" and all rules from "mychecks3" rule set.

```
ptc_shell> disable_rule {CLK_0001 mychecks3}
```

SEE ALSO

analyze_design(2)
create_rule(2)
create_ruleset(2)
enable_rule(2)
get_rules(2)
remove_rule(2)

echo

Echos arguments to standard output.

SYNTAX

```
string echo
   [-n]
   [arguments]
```

Data Types

```
   arguments      string
```

ARGUMENTS

-n

Suppresses the new line. By default, **echo** adds a new line.

arguments

Specifies the arguments to be printed.

DESCRIPTION

The **echo** command prints out the value of the given arguments. Each of the arguments are separated by a space. The line is normally terminated with a new line, but if the **-n** option is specified, multiple **echo** command outputs are printed onto the same output line.

The **echo** command is used to print out the value of variables and expressions in addition to text strings. The output from **echo** can be redirected using the **>** and **>>** operators.

EXAMPLES

The following are examples of using the **echo** command:

```
prompt> echo
"Running version" $sh_product_version
Running version v3.0a

prompt> echo -n "Printing to" [expr (3 - 2)] > foo
prompt> echo " line." >> foo
prompt> sh cat foo
Printing to 1 line.
```

SEE ALSO

sh(2)

enable_rule

Enables specified built-in or user-defined rules.

SYNTAX

```
Boolean enable_rule
   rule_list
```

Data Types

```
   rule_list     list
```

ARGUMENTS

rule_list

The list of rules or rule sets to enable. If a rule set is specified, all rules from that rule set are enabled.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Enables specified rules if they are currently disabled. During **analyze_design**, violations are created for disabled rules. Note that some built-in rules are disabled by default, and must be enabled if you want them to be checked. User-defined rules are all enabled when they are created.

EXAMPLES

The following example enables rule "CLK_0001" and all rules from "mychecks3" rule set.

```
ptc_shell> enable_rule {CLK_0001 mychecks3}
```

SEE ALSO

analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
get_rules(2)
remove_rule(2)

error_info

Prints extended information on errors from the last command.

SYNTAX

```
string error_info
```

ARGUMENTS

This **error_info** command has no arguments.

DESCRIPTION

The **error_info** command is used to display information after an error has occurred. Tcl collects information showing the call stack of commands and procedures. When an error occurs, the **error_info** command can help you to focus on the exact line in a block that caused the error.

EXAMPLES

This example shows how **error_info** can be used to trace an error. The error is that the iterator variable "s" is not dereferenced in the 'if' statement. It should be '$s == "a" '.

```
prompt> foreach s $my_list {
?         if { s == "a" } {
?           echo "Found 'a'!"
?         }
?       }
Error: syntax error in expression " s == "a" "
       Use error_info for more info. (CMD-013)
shell> error_info
```

```
Extended error info:
syntax error in expression " s == "a" "
    while executing
"if { s == a } {
     echo "Found 'a'"
}"
    ("foreach" body line 2)
    invoked from within
"foreach s [list a b c] {
   if { s == a } {
     echo "Found 'a'"
}
}"
 -- End Extended Error Info
```

exit

Terminates the application.

SYNTAX

```
integer exit
   [exit_code]
```

Data Types

```
   exit_code      integer
```

ARGUMENTS

exit_code

Specifies the return code to the operating system. The default value is 0.

DESCRIPTION

This command exits from the application. You have the option to specify a code to return to the operating system.

EXAMPLES

The following example exits the current session and returns the code 5 to the operating system. At a UNIX operating system prompt, verify (**echo**) the return code as shown.

```
prompt> exit 5
```

```
% echo $status
5
```

SEE ALSO

quit(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]
```

Data Types

```
   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 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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 if 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 if it is valid for the object class.

For a complete description of conditional expressions, see the *PrimeTime User Guide*.

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 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 make the regular expression search case-insensitive by using the **-nocase** option.

EXAMPLES

The following example creates a collection of only hierarchical cells.

```
ptc_shell> set a [filter_collection [get_cells *] \
"is_hierarchical == true"]
{"Adder1", "Adder2"}
```

The following example creates a collection of all non-mode cell timing_arc objects in the current design.

```
ptc_shell> set b [filter_collection \
[get_timing_arcs -of_objects [get_cells *]] \
"undefined(mode)"]
```

SEE ALSO

collections(2)
regexp(2)

foreach_in_collection

Iterates over the elements of a collection.

SYNTAX

```
string foreach_in_collection
   itr_var
   collections
   body
```

Data Types

```
   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

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **foreach_in_collection** command is used to iterate over each element in a collection. You cannot use the Tcl-supplied **foreach** 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. Note that **foreach_in_collection** 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**.

Note that 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 ends 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 needed 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, though 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.

```
ptc_shell> 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)

get_app_var

Gets the value of an application variable.

SYNTAX

```
string get_app_var
   [-default | -details | -list]
   [-only_changed_vars]
   var
```

Data Types

```
   var     string
```

ARGUMENTS

-default

Gets the default value.

-details

Gets additional variable information.

-list

Returns a list of variables matching the pattern. When this option is used, then the *var* argument is interpreted as a pattern instead of a variable name.

-only_changed_vars

Returns only the variables matching the pattern that are not set to their default values, when specified with **-list**.

var

Specifies the application variable to get.

DESCRIPTION

The **get_app_var** command returns the value of an application variable.

There are four legal forms for this command:

- `get_app_var <var>`
  `Returns the current value of the variable.`

- `get_app_var <var> -default`
  `Returns the default value of the variable.`

- `get_app_var <var> -details`

Returns more detailed information about the variable. See below for details.

- get_app_var -list [-only_changed_vars] <*pattern*>

Returns a list of variables matching the pattern. If *-only_changed_vars* is specified, then only variables that are changed from their default values are returned.

In all cases, if the specified variable is not an application variable, then a Tcl error is returned, unless the application variable **sh_allow_tcl_with_set_app_var** is set to true. See the **sh_allow_tcl_with_set_app_var** man page for details.

When **-details** is specified, the return value is a Tcl list that is suitable as input to the Tcl **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key, and each even-numbered element in the list is the value of the previous key.

The supported keys are as follows:

name

This key contains the name of the variable. This key is always present.

value

This key contains the current value of the variable. This key is always present.

default

This key contains the default value of the variable. This key is always present.

help

This key contains the help string for the variable. This key is always present, but sometimes the value is empty.

type

This key contains the type of the application variable. Legal values of for this key are: string, bool, int, real. This key is always present.

constraint

This key describes additional constraints placed on this variable. Legal values for this key are: none, list, range. This key is always present.

min

This key contains the min value of the application variable. This key is present if the constraint is range. The value of this key may be the empty string, in which case the variable only has a max value constraint.

max

This key contains the max value of the application variable. This key is present if the constraint is "range". The value of this key may be the empty string, in which case the variable only has a min value constraint.

list

This key contains the list of legal values for the application variable. This key is present if the constraint is "list".

EXAMPLES

The following are examples of the **get_app_var** command:

```
prompt> get_app_var sh_enable_page_mode
1

prompt> get_app_var sh_enable_page_mode -default
false

foreach {key val} [get_app_var sh_enable_page_mode -details] {   echo "$key: $val"
}
=>  name: sh_enable_page_mode
    value: 1
    default: false
    help: Displays long reports one page at a time
    type: bool
    constraint: none

prompt> get_app_var -list sh_*message
sh_new_variable_message
```

SEE ALSO

```
report_app_var(2)
set_app_var(2)
write_app_var(2)
```

get_attribute

Retrieves the value of an attribute on an object.

SYNTAX

```
string get_attribute [-class class_name]
[-quiet]
[-value_list]
object_spec
attr_name

string  class_name
string  object_spec or
collection object_spec
string  attr_name
```

ARGUMENTS

-class *class_name*

Specifies the class name of *object_spec*, if *object_spec* is a name. Valid values for *class_name* are design, cell, pin, port, net, lib, lib_cell, lib_pin, timing_arc, lib_timing_arc, clock, scenario, input_delay, output_delay, exception, exception_group, clock_group, clock_group_group, rule, ruleset, and rule_violation. You must use this option if *object_spec* is a name.

-quiet

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

-value_list

Normally the return value of the **get_attribute** command is a string if there is only a single object, and a list if multiple objects are specified. This option indicates that the return value should be a list, even if there is only a single object specified to retrieve the attribute.

object_spec

Specifies a single object from which to get the attribute value. *object_spec* must be either a collection of exactly one object, or a name which 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

This command is available only if you invoke the pt_shell with the **-constraints** option.

Retrieves the value of an attribute on an object. The object is either a collection of exactly one object, or a 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 command retrieves the application attribute **full_name** and **direction** of a port and creates a simple report.

```
ptc_shell_shell> foreach_in_collection sel [get_ports *] {
?           echo -n "Port '[get_attribute $sel full_name]' is an "
?           echo "'[get_attribute $sel direction]' port "
?         }
Port 'A' is an 'in' port
Port 'B' is an 'in' port
Port 'CLK' is an 'in' port
Port 'RESET' is an 'in' port
Port 'QA' is an 'out' port
Port 'QB' is an 'out' port
Port 'CO' is an 'out' port
```

SEE ALSO

collections(2)
foreach_in_collection(2)
list_attributes(2)
report_attribute(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]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-filter expression]
   patterns | -of_objects objects
```

Data Types

```
   expression      string
   objects         list
   patterns        list
```

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".

-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. **-regexp** and **-exact** 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. For use when searching for objects that contain the * and ? wildcard characters. **-exact** and **-regexp** 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, a pin collection, a named net or a net collection. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both. In addition, you cannot use **-hierarchical** with **-of_objects**.

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. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 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; 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.

```
ptc_shell> 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.

```
ptc_shell> set pinsel [get_pins o*/CP]
{"o_reg1/CP", "o_reg2/CP"}
ptc_shell> query_objects [get_cells -of_objects $pinsel]
{"o_reg1", "o_reg2"}
```

The following example removes the wire load model from cells i1 and i2.

```
ptc_shell> 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_clock_crossing_points

Gets a collection of clock-crossing endpoints between the given launch and the capture clocks.

SYNTAX

```
string get_clock_crossing_points
   [-from from_clock]
   [-to to_clock]
   [-valid]
   [-quiet]
```

Data Types

```
   from_clock    collection of one object
   to_clock      collection of one object
   valid         Boolean
   quiet         Boolean
```

ARGUMENTS

-from *from_clock*

Launch clock for getting endpoints.

-to *to_clock*

Capture clock for getting endpoints.

-valid

Does not include endpoints that only have false paths reaching them.

-quiet

Does not issue an error if no endpoints are found.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Gets a collection of endpoints where the launch clock is the *from_clock* and the capture clock is *to_clock*. By default, all such endpoints are included, even though the paths between launch clock and capture clock are all **false**, or if there is a clock groups relationship between the clocks.

To obtain only non-false path endpoints, use the *-valid* option. To not report any error if no such endpoints could be found, use the *-quiet* option.

EXAMPLES

The following example shows a simple usage:

```
ptc_shell> get_clock_crossing_points -from CLK1 -to CLK2
{"ffa/D", "ffb/D"}
```

SEE ALSO

analyze_paths(2)
report_clock_crossing(2)

get_clock_group_groups

Creates a collection of the clock_group_groups (groups of a clock_group) in a clock_group in the current scenario. You can assign these clock_group_groups to a variable or pass them into another command.

SYNTAX

```
collection get_clock_group_groups
   [-quiet]
   clock_group
```

Data Types

```
   clock_group     string
```

ARGUMENTS

clock_group

The clock_group object for which the clock_group_group objects need to be returned. Only one clock_group object is allowed in this command.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_clock_group_groups** command creates a collection of clock_group_group objects of a specific clock_group object in the current scenario. You can assign these clock_group_groups to a variable or pass them into another command.

The command returns a collection if the clock_group object specified exists, else returns an empty string.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

In the following example the variable cg is set to a clock_group object. It then sets the variable cg_groups with the clock_group_group objects of that clock_group.

```
ptc_shell> set cg_groups [get_clock_group_groups $cg]
ptc_shell>
```

SEE ALSO

collections(2)
set_clock_groups(2)
collection_result_display_limit(3)

get_clock_groups

Creates a collection of clock_groups in the current scenario. You can assign these clock_groups to a variable or pass them into another command.

SYNTAX

```
collection get_clock_groups
   [-quiet]
   [-of_objects objects]
   [-filter expression]
   patterns
```

Data Types

```
   objects         list
   expression      string
   patterns        list
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-of_objects *objects*

Creates a collection of clock_groups containing the specified objects. Each object is a named clock or clock collection. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both.

-filter *expression*

Filters the collection with *expression*. For any clock_group that match the pattern or the of_objects criteria, the expression is evaluated based on the clock_group's attributes. If the expression evaluates to true, the clock_group is included in the result.

patterns

Matches clock_group names against patterns. Patterns can include the wildcard characters "*" and "?". *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_clock_groups** command creates a collection of clock_groups in the current scenario that match certain criteria. You can assign these clock_groups to a variable or pass them into another command.

The command returns a collection if any clock_group object matches the pattern or the -of_objects specifiers and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_clock_groups** 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_clock_groups** result to a variable.

When issued from the command prompt, **get_clock_groups** 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_clock_groups** provides a fast, simple way to display clock_groups 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_clock_groups** as an argument to **query_objects**. Note that the name shown for a clock_group is either the one given during this clock_group's specification and if not then its generated by the tool.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example sets the variable cg to a collection of asynchronous clock_groups which contains the clock clk1.

```
ptc_shell> set cg [get_clock_groups -of_objects {clk1} -filter "clock_group_type==asynchronous"]
```

SEE ALSO

```
collections(2)
set_clock_groups(2)
collection_result_display_limit(3)
```

get_clock_network_objects

Returns a collection of objects that belong or relate to the direct clock network.

SYNTAX

```
collection get_clock_network_objects
   -type object_type
   [clock_list]
```

Data Types

```
   object_type          string
   clock_list           list
```

ARGUMENTS

-type *object_type*

Specifies the type of the clock network objects to be returned.

clock_list

Specifies the clock domains of which the clock network objects are returned. By default, the command returns all clock network objects.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command returns a collection of clock network objects of certain type (specified by the *object_type* option) that belong or relate to one or several clock domains (specified by the *clock_list* option), or all clock domains if the *clock_list* option is not specified.

A clock network is a special logic part of the design that propagates the clocks from the clock sources to the clock pins of latches, flip-flops (that function as anything but propagating clocks) or black-box IPs. The

propagation also stops at design output ports, dangling pins or nets, or the sources of other clocks. The **get_clock_network_objects** command retrieves certain types of objects from the direct clock network (including the latches, flip-flops and black box IPs driven by the clock network). If the specified clock is a generated clock, the propagation does not go backward to the clock network of its master clock. If the specified clock is a master clock, the propagation does not go forward to the clock network of its generated clocks either.

The *object_type* option can be one of the following:

cell

Cells that belong to the clock network, including noninverting or inverting buffers, combinational cells, library defined clock gating cells, and so on.

register

Latches, flip-flops or black box IPs, such as memory cells, that are driven by the clock network.

net

Nets that connect the clock network cells to other clock network cells, to the driven registers, or to the I/O ports of design.

pin

Pins and ports that are connected with the clock network nets. Note that the clock pins of the driven registers are included.

clock_gating_output

Output pins of clock gating cells, either defined by the library, inserted by Power Compiler, automatically inferred by constraint consistency, or manually set by you. The results are consistent with those of **report_clock_gating_checks**, and can be affected by other clock gating check related commands or variables.

EXAMPLES

The following command returns a collection of clock network pins of all clock domains in the design, not including pins of the clock gating networks.

```
ptc_shell> get_clock_network_objects -type pin
```

SEE ALSO

analyze_clock_networks(2)
create_clock(2)
create_generated_clock(2)
get_clocks(2)

```
get_generated_clocks(2)
remove_clock(2)
remove_generated_clock(2)
report_clock(2)
remove_clock_gating_check(2)
remove_disable_clock_gating_check(2)
report_clock_gating_check(2)
set_clock_gating_check(2)
set_disable_clock_gating_check(2)
timing_disable_clock_gating_checks(2)
```

get_clocks

Creates a collection of clocks from the current design. You can assign these clocks to a variable or pass them into another command.

SYNTAX

```
collection get_clocks
   [-quiet]
   [-regexp]
   [-nocase]
   [-filter expression]
   patterns
```

Data Types

```
   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 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.

-nocase

When combined with **-regexp**, makes matches case-insensitive. You can use **-nocase** only when you also use **-regexp**.

-filter *expression*

Filters the collection with *expression*. For any clocks that match *patterns*, the expression is evaluated based on the clock's attributes. If the expression evaluates to true, the clock is included in the result.

patterns

Matches clock names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_clocks** command creates a collection of clocks in the current design that match certain criteria. The command returns a collection if any clocks match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_clocks** 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_clocks** result to a variable.

When issued from the command prompt, **get_clocks** 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_clocks** provides a fast, simple way to display clocks 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_clocks** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page. In addition, see the man page for **all_clocks**, which also creates a collection of clocks.

EXAMPLES

The following example applies the **set_max_time_borrow** command to all clocks in the design matching "PHI*".

```
ptc_shell> set_max_time_borrow 0 [get_clocks "PHI*"]
```

The following example sets the variable clock_list to a collection of clocks that have period of 15.

```
ptc_shell> set clock_list [get_clocks * -filter "period==15"]
```

SEE ALSO

```
all_clocks(2)
collections(2)
```

```
    create_clock(2)
    query_objects(2)
    report_clock(2)
    collection_result_display_limit(3)
```

get_command_option_values

Queries current or default option values.

SYNTAX

```
get_command_option_values
   [-default | -current]

   -command command_name
```

Data Types

```
   command_name      string
```

ARGUMENTS

-default

Gets the default option values, if available.

-current

Gets the current option values, if available.

-command *command_name*

Gets the option values for this command.

DESCRIPTION

This command attempts to query a default or current value for each option (of the command) that has default and/or current-value-tracking enabled. Details of how the option value is queried depend on whether one of the **-current** or **-default** options is specified (see below).

A "Tcl array set compatible" (possibly empty) list of option names and values is returned as the Tcl result. The even-numbered entries in the list are the names of options that were enabled for default-value-tracking or current-value-tracking and had at least one of these values set to a not-undefined value). Each odd-numbered entry in the list is the default or current value of the option name preceding it in the list.

Any options that were not enabled for either default-value-tracking nor current-value-tracking are omitted from the output list. Similarly, options that were enabled for default-value-tracking or current-value-tracking, but for which no (not-undefined) default or current value is set, are omitted from the result list.

If neither **-current** nor **-default** is specified, then for each command option that has either default-value-tracking or current-value-tracking (or both) enabled, the value returned is as follows:

- The current value is returned if current-value-tracking is enabled and a (not-undefined) current value has been set;
- Otherwise the default value is returned if default-value-tracking is enabled and a (not-undefined) default value has been set;
- Otherwise the name and value pair for the option is not included in the result list.

If **-current** is specified, the value returned for an option is the current value if current-value-tracking is enabled, and a (not-undefined) current value has been set; otherwise the name and value pair for the option is omitted from the result list.

If **-default** is specified, the value returned for an option is the default value if default-value-tracking is enabled, and a (not-undefined) default value has been set; otherwise the name and value pair for the option is omitted from the result list.

The result list from **get_command_option_values** includes option values of both dash options and positional options (assuming that both kinds of options of a command have been enabled for value-tracking).

The command issues a Tcl error in a variety of situations, such as if an invalid command name was passed in with **-command**.

EXAMPLES

The following example shows the use of **get_command_option_values**:

```
prompt> test -opt1 10 -opt2 20
1

prompt> get_command_option_values -command test
-bar1 10 -bar2 20
```

SEE ALSO

```
    preview(2)
    set_command_option_value(2)
```

get_defined_attributes

Returns attribute names currently defined for the specified class.

SYNTAX

```
string get_defined_attributes
   -class class_name
   -return_classes
   [-details]
   [-application]
   [-user]
   [attr_pattern]
```

Data Types

```
   class_name          string
   attribute_names     list
   data_type           string
```

ARGUMENTS

-class *class_name*

Specifies the class for which to get the attributes. Valid classes are application defined. Note legal with the *-return_classes* option.

-return_classes

Return the set of application defined classes. Not legal with the *-class* option.

-details

Gets additional information on a single attribute.

-application

Returns application-defined attributes only. This option is mutually exclusive with the *-user* option.

-user

Returns user-defined attributes only. This option is mutually exclusive with the *-application* option.

attr_patterns

Specifies a list of attribute names or glob style patterns to check on the existences of.

DESCRIPTION

This command returns a list of attribute names that are defined for the specified class.

There are three legals forms for this command:

- `get_defined_attributess -return_classes`

Returns the application defined object classes

- get_defined_attributes -class cell [<pattern>] Returns all the attributes matching the optional pattern
- get_defined_attributes -class cell -details <attrName> Returns more detailed information about the attribute. See below for details

When **-details** is specified, the return value is a Tcl list that is suitable as input to the Tcl **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key, and each even-numbered element in the list is the value of the previous key.

The supported keys are as follows:

name

This key contains the name of the attribute

infor

The short help defined for the attribbute

type

The type of value the attribute storesn

settable

The value is settable with the set_attribute command.

application

The value is 1 if the attribute is application defined, else 0.

subscripted

The value is 1 if the attribute is subscripted, else 0.

min_value

Not always present. If present value is the minimum allowed value.

max_value

Not always present. If present value is the maximum allowed value.

allowd_values

Not always present. If present value is the set of allowed values

allowed_subscripts

Present if the attribute is subscripted. Contains list of legal subcripts for the attribute if the attribute uses global subscripts. If empty, then the subscripts vary per object.

smart_subscripts

Present if the attribute is subscripted. Contains the list of supported smart subscripts if the attribute supports smart subscripts.

EXAMPLES

The following are examples of the **get_defined_attributes** command:

```
prompt> get_defined_attributes -class cell
name full_name object_class ....

prompt> get_defined_attributes -class cell *name*
name full_name reference_name ...

prompt> get_defined_attributes -class cell name
name name info {The simple name of the cell} type string application 1
settable 0 subscripted 0
```

SEE ALSO

```
get_attribute(2)
help_attributes(2)
```

get_defined_commands

Get information on defined commands and groups.

SYNTAX

```
string get_defined_commands [-details]
   [-groups]
   [pattern]

string pattern
```

ARGUMENTS

-details

Get detailed information on specific command or group.

-groups

Search groups rather than commands

pattern

Return commands or groups matching pattern. The default value of this argument is "*".

DESCRIPTION

The **get_defined_commands** gets information about defined commands and command groups. By default the command returns a list of commands that match the specified pattern.

When **-details** is specified, the return value is a list that is suitable as input to the **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key name, and each even-numbered element in the list os the value of the previous key. The **-details** option is only legal if the pattern matches exactly one command or group.

When **-group** is specified with **-details**, the supported keys are as follows:

name

This key contains the name of the group.

info

This key contains the short help for the group.

commands

This key contains the commands in the group

When **-details** is used with a command, the supported keys are as follows:

name

This key contains the name of the command.

info

This key contains the short help for the command.

groups

This key contains the group names that this command belongs to.

options

This key contains the options defined for the command. The value is a list.

return

This key contains the return type for the command.

Each element in the *options* list also follows the key value pattern. The set of available keys for options are as follows:

name

This key contains the name of the option.

info

This key contains the short help for the option.

value_info

This key contains the short help string for the value

type

The type of the option.

required

The value will be 0 or 1 depending if the option is optional or required.

is_list

Will be 1 if the option requires a list.

list_length

If a list contains the list length constraint. One of any, even, odd, non_empty or a number of elements.

allowed_values

The allowed values if the option has specified allowed values.

min_value

The minimum allowed value if the option has specified one.

max_value

The maximum allowed value if the option has specified one.

EXAMPLES

```
prompt> get_defined_commands *collection
add_to_collection append_to_collection copy_collection filter_collection
foreach_in_collection index_collection sort_collection
prompt> get_defined_commands -details sort_collection
name sort_collection info {Create a sorted copy of the collection}
groups {} options {{name -descending info {Sort in descending order}
value_info {} type boolean required 1 is_list 0} {name -dictionary info
{Sort strings dictionary order.} value_info {} type boolean required 1
is_list 0} {name collection info {Collection to sort} value_info
collection type string required 0 is_list 0} {name criteria info {Sort
criteria - list of attributes} value_info criteria type list required 0
is_list 1 list_length non_empty}}
```

SEE ALSO

```
help(2)
man(2)
```

get_designs

Creates a collection of one or more designs loaded in constraints consistency. You can assign these designs to a variable or pass them into another command.

SYNTAX

```
collection get_designs
[-hierarchical]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-filter expression]
patterns
```

Data Types

expression	string
patterns	list

ARGUMENTS

-hierarchical

Searches for designs inferred by the design hierarchy relative to the current instance. The full name of the object at a particular level must match the patterns. The use of 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. Also, it 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

When combined with the **-regexp** option, this option makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-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.

-filter *expression*

Filters the collection with *expression*. For any designs that match *patterns*, the expression is evaluated based on the design's attributes. If the expression evaluates to true, the design is included in the result.

patterns

Matches design 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 design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_designs** command creates a collection of designs from those currently loaded into constraint consistency that match certain criteria. This command returns a collection if any designs match the *patterns* argument and pass the filter (if specified). If no objects matched your 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 *patterns* and filter *expression*. 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_designs** 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_designs** result to a variable.

When issued from the command prompt, **get_designs** 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_designs** provides a fast, simple way to display designs 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_designs** 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 designs that begin with "mpu." Although the output looks like a list, it is just a display. A complete listing of designs is available using the **list_designs** command.

```
ptc_shell> get_designs mpu*
{"mpu_0_0", "mpu_0_1", "mpu_1_0", "mpu_1_1"}
```

The following example shows that, given a collection of designs, you can remove those designs.

```
ptc_shell> remove_design [get_designs mpu*]
Removing design mpu_0_0...
Removing design mpu_0_1...
Removing design mpu_1_0...
Removing design mpu_1_1...
```

SEE ALSO

collections(2)
filter_collection(2)
list_designs(2)
query_objects(2)
regexp(2)
remove_design(2)
collection_result_display_limit(3)

get_exception_groups

Creates a collection of exception_groups from a collection of exceptions. You can assign the result collection to a variable or pass them into another command.

SYNTAX

```
collection get_exception_groups
   [-filter expression]
   collection1
```

Data Types

```
   expression       string
   collection1      collection
```

ARGUMENTS

collection

Specifies the collection of exceptions to be filtered.

-filter *expression*

Filters the collection with the *expression* option. For any exception_groups that match the *patterns* option, the expression is evaluated based on the exception_group's attributes. If the expression evaluates to true, the exception_group is included in the result. If this option is not given, then all exception_groups in the *collection* is added to the result collection.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_exception_groups** command creates a collection of exception_groups from the *collection1* collection. The command returns a collection containing all exception_groups matching the *expression* argument of the -filter option.

You can use the **get_exception_groups** command at the command prompt, or you can nest it as an argument to another command. In addition, you can assign the **get_exception_groups** result to a variable.

For information about getting collection of exceptions, see the **get_exceptions** man page.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following command sequence first get a collection of exceptions with from pins "**a***". The result collection is assigned to a variable named **exp_obj**.

The second command then, from **exp_obj**, gather a collection of exception_groups with **type** equal to "through". The result is assigned to a variable named **through_groups_clct**.

```
ptc_shell> set exp_obj [get_exceptions -from a*]
ptc_shell> set through_groups_clct [get_exception_groups $exp_obj -filter "type==through"]
```

SEE ALSO

```
get_exceptions(2)
collection_result_display_limit(3)
list_attributes(2)
report_attribute(2)
exception_group_attributes(3)
```

get_exceptions

Creates a collection of exceptions in the current scenario. You can assign these exceptions to a variable or pass them into another command.

SYNTAX

```
collection get_exceptions [-quiet]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-filter expression]
```

Data Types

```
expression     string
```

ARGUMENTS

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, and nets through which the paths must pass that are to be reset. Nets are interpreted to imply the net segment's local driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than one time in one command invocation. For a discussion of the use of multiple **-through** options, see the DESCRIPTION section.

-rise_through ***rise_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the through points. You can specify **-rise_through** more than one time in one command invocation. For a discussion of the use of multiple **-through** options, see the DESCRIPTION section.

-fall_through ***fall_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the through points. You can specify **-fall_through** more than one time in one command invocation. For a discussion of the use of multiple **-through** options, see the DESCRIPTION section.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter ***expression***

Filters the collection with *expression*. For any exceptions that match the from/through/to criteria, the expression is evaluated based on the exception's attributes. If the expression evaluates to true, the exception is included in the result.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_exceptions** command creates a collection of exceptions in the current scenario that match certain criteria. You can assign these exceptions to a variable or pass them into another command.

The command returns a collection if any exceptions match the from/through/to specifiers and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_exceptions** 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_exceptions** result to a variable.

When issued from the command prompt, **get_exceptions** 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_exceptions** provides a fast, simple way to display exceptions 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_exceptions** as an argument to **query_objects**. Note that the name shown for an exception is a brief id for printing only. You cannot search for exceptions by this name.

For information about collections and the querying of objects, see the **collections** man page. In addition, see the man page for **all_exceptions**, which also creates a collection of exceptions.

EXAMPLES

The following example sets the variable e1 to a collection of false_path exceptions with OUT1 as a to_pin.

```
ptc_shell> set e1 [get_exceptions -to OUT1 -filter "type==false_path"]
```

SEE ALSO

all_exceptions(2)
collections(2)
set_false_path(2)
set_multicycle_path(2)
set_max_delay(2)
set_min_delay(2)
collection_result_display_limit(3)

get_generated_clocks

Creates a collection of generated clocks.

SYNTAX

```
collection get_generated_clocks
   [-quiet]
   [-regexp]
   [-nocase]
   [-filter expression]
   patterns
```

Data Types

```
   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 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.

-nocase

When combined with the *-regexp* option, makes matches case-insensitive. You can use the *-nocase* option only when you also use the *-regexp* option.

-filter *expression*

Filters the collection with the *expression* option. For any generated clocks that match the *patterns* option, the expression is evaluated based on the generated clock's attributes. If the expression evaluates to true, the generated clock is included in the result.

patterns

Matches generated clock names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_generated_clocks** command creates a collection of generated clocks from the current design that match certain criteria. The command returns a collection if any generated clocks match the *patterns* option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

To create a generated clock in the design, use the **create_generated_clock** command. To remove a generated clock from the design, use the **remove_generated_clock** command. To show information about clocks and generated clocks in the design, use the **report_clock** command.

You can use the **get_generated_clocks** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_generated_clocks** command result to a variable.

When issued from the command prompt, the **get_generated_clocks** command behaves as though the **query_objects** command 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 **collection_result_display_limit** variable.

The "implicit query" property of the **get_generated_clocks** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_generated_clocks** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following command applies the **set_clock_latency** command on all generated clocks in the design matching the "GEN*".

```
ptc_shell> set_clock_latency 2.0 [get_generated_clocks "GEN*"]
```

The following command removes all the generated clocks in the design matching "GEN1*".

```
ptc_shell> remove_generated_clock [get_generated_clocks "GEN1*"]
```

SEE ALSO

collections(2)
create_generated_clock(2)
query_objects(2)
remove_generated_clock(2)
collection_result_display_limit(3)

get_input_delays

Creates a collection of input_delays in the current scenario. You can assign these input_delays to a variable or pass them into another command.

SYNTAX

```
collection get_input_delays
   [-quiet]
   [-filter expression]
   -of_objects objects
```

Data Types

```
   expression    string
   objects       list
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with *expression*. For any input_delays on *objects*, the expression is evaluated based on the input_delay's attributes. If the expression evaluates to true, the input_delay is included in the result.

-of_objects *objects*

Creates a collection of input_delays defined on the specified objects. Each object is either a named pin, a pin collection, a named port or port collection.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_input_delays** command creates a collection of input_delays matching *patterns* in the current scenario and which pass the filter (if specified). If no objects match the criteria, the empty string is returned.

EXAMPLES

This example sets variable inDel to the collection of input_delays defined on ports IN*.

```
ptc_shell> set inDel [get_input_delays -of [get_ports IN*]]
```

SEE ALSO

collections(2)
set_input_delay(2)

get_input_unit

Returns a string representing the input unit for one unit type.

SYNTAX

```
string get_input_unit
   -type unit_type
   [-numeric]
```

Data Types

```
   unit_type     string
```

ARGUMENTS

-type *unit_type*

Specifies the type of quantity, which must be one of the values: "time", "resistance", "capacitance", "voltage", "current", or "power".

-numeric

If specified, returns the value as a number representing the unit value in terms of the base unit for that unit type. Base units are Seconds, Ohms, Farads, Volts, Amperes, and Watts.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command returns a string representing the input unit for one unit type. The tool maintains separate units for input and output. There are input unit values for time, resistance, capacitance, voltage, current, and power.

EXAMPLES

The following example shows how to get the input unit for time. With the **-numeric** option, the value is returned in terms of the base unit for that quantity type (seconds, for time).

```
ptc_shell> get_input_unit -type time
1.00ps
ptc_shell> get_input_unit -type time -numeric
1e-12
```

SEE ALSO

set_input_units(2)

get_lib_cells

Creates a collection of library cells from libraries loaded into constraint consistency. You can assign these library cells to a variable or pass them into another command.

SYNTAX

```
collection get_lib_cells [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns | -of_objects objects
```

Data Types

```
   expression     string
   objects        list
   patterns       list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any library cells that match *patterns* (or *objects*), the expression is evaluated based on the library cell's attributes. If the expression evaluates to true, the library 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. **-regexp** and **-exact** 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. **-exact** and **-regexp** are mutually exclusive.

-of_objects ***objects***

Creates a collection of library cells that are referenced by the specified cells or own the specified library pins. In this case, each object is either a named library pin or netlist cell, or a library pin collection or a netlist cell collection. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both.

patterns

Matches library 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 lib_cell. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_lib_cells** command creates a collection of library cells from libraries currently loaded into constraint consistency that match certain criteria. The command returns a collection if any library cells match the *patterns* or *objects* 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*. 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_lib_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_lib_cells** result to a variable.

When issued from the command prompt, **get_lib_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_lib_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_lib_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 all library cells that are in the misc_cmos library and begin with AN2. Although the output looks like a list, it is just a display.

```
ptc_shell> get_lib_cells misc_cmos/AN2*
{"misc_cmos/AN2", "misc_cmos/AN2P"}
```

The following example shows one way to find out the library cell used by a particular cell.

```
ptc_shell> get_lib_cells -of_objects [get_cells o_reg1]
{"misc_cmos/FD2"}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_cells(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)
```

get_lib_pins

Creates a collection of library cell pins from libraries loaded in constraint consistency. You can assign these library cell pins to a variable or pass them into another command.

SYNTAX

```
collection get_lib_pins [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns | -of_objects objects
```

Data Types

```
   expression    string
   objects       list
   patterns      list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any library cell pins that match *patterns* (or *objects*), the expression is evaluated based on the library cell pin's attributes. If the expression evaluates to true, the lib_pin 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. **-regexp** and **-exact** 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. **-exact** and **-regexp** are mutually exclusive.

-of_objects ***objects***

Creates a collection of library cell pins referenced by the specified netlist pins or owned by the specified library cells. In this case, each object is either a named library cell or netlist pin, or a library cell collection or netlist pin collection. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both.

patterns

Matches library cell 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 lib_pin. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_lib_pins** command creates a collection of library cell pins from libraries currently loaded into constraint consistency that match certain criteria. The command returns a collection if any library cell pins match the *patterns* or *objects* and pass the filter (if specified). If no objects matched 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*. 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_lib_pins** 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_lib_pins** result to a variable.

When issued from the command prompt, **get_lib_pins** 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_lib_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_lib_pins** 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 all pins of the AN2 library cell in the misc_cmos library. Although the output looks like a list, it is just a display.

```
ptc_shell> [get_lib_pins misc_cmos/AN2/*
{"misc_cmos/AN2/A", "misc_cmos/AN2/B", "misc_cmos/AN2/Z"}
```

The following example shows one way to find out how the library pin is used by a particular pin in the netlist.

```
ptc_shell> get_lib_pins -of_objects o_reg1/Q
{"misc_cmos/FD2/Q"}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_libs(2)
get_lib_cells(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)
```

get_lib_timing_arcs

Creates a collection of library arcs for custom reporting and other processing. You can assign these library arcs to a variable and get the required attribute for further processing.

SYNTAX

```
string get_lib_timing_arcs [-to to_list]
   [-from from_list]
   [-of_objects object_list]
   [-filter expression]
   [-quiet]
```

Data Types

```
   to_list          list
   from_list        list
   object_list      list
   expression       string
```

ARGUMENTS

-to *to_list*

Specifies the "to" library pins, or ports. All backward library arcs from the specified library pins or ports are considered.

-from *from_list*

Specifies the "from" library pins, or ports. All forward library arcs from the specified library pins or ports are considered.

-of_objects *object_list*

Specifies library cells or timing arcs. If a library cell is specified, all library cell arcs of that cell are considered. If a timing arc collection is given in the object list, the corresponding library timing arc is considered.

-filter *expression*

Specifies the filter expression. A filter expression is a string that comprises a series of logical expressions describing a set of constraints you want to place on the collection of library arcs. Each

subexpression of a filter expression is a relation contrasting an attribute name with a value, by means of an operator.

-quiet

Specifies that all messages are to be suppressed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Creates a collection of library arcs for custom reporting and other processing. You can assign these library arcs to a variable and get the required attribute for further processing. Use the **foreach_in_collection** command to iterate among the library arcs in the collection. You can use the **get_attribute** command to obtain information about the paths. You can also use the filter expression to filter the library arcs that satisfy the specified conditions. The following attributes are supported on library timing arcs:

from_lib_pin
is_disabled
is_user_disabled
mode
object_class
sdf_cond
sense
to_lib_pin

One attribute of a library timing arc is the **from_lib_pin**, which is the library pin from which the timing arc begins. In the same way, **to_lib_pin** is the library pin to which the timing arc ends. To get more information on **from_lib_pin** and **to_lib_pin**, use the **get_attributes** command. The **object_class** attribute holds the class name of library timing arcs: **lib_timing_arc**.

See the **collections** and **foreach_in_collection** man pages for more information.

EXAMPLES

The following procedure is an example of how the collection returned from an invocation of **get_lib_timing_arcs** can be used to provide useful and readable cell level arc information.

```
proc show_lib_arcs {args} {
  set lib_arcs [eval [concat get_lib_timing_arcs $args]]
  echo [format "%15s    %-15s  %18s" "from_lib_pin" "to_lib_pin" \
            "sense"]
  echo [format "%15s    %-15s  %18s" "------------" "----------" \
            "-----"]

  foreach_in_collection lib_arc $lib_arcs {
    set fpin [get_attribute $lib_arc from_lib_pin]
    set tpin [get_attribute $lib_arc to_lib_pin]
    set sense [get_attribute $lib_arc sense]
```

```
        set from_lib_pin_name [get_attribute $fpin base_name]
        set to_lib_pin_name [get_attribute $tpin base_name]
        echo [format "%15s -> %-15s  %18s" \
            $from_lib_pin_name $to_lib_pin_name \
            $sense]
      }
    }
```

ptc_shell> **show_lib_arc -of_objects [get_timing_arcs -of_objects ffa]**

```
from_lib_pin    to_lib_pin                    sense
------------    ----------                    -----
           CP -> D                    setup_clk_rise
           CP -> D                     hold_clk_rise
           CP -> Q                       rising_edge
           CP -> QN                      rising_edge
           CD -> Q                         clear_low
           CD -> QN                       preset_low
```

ptc_shell> **show_lib_arc -of_objects mylib/AN2**

```
from_lib_pin    to_lib_pin                    sense
------------    ----------                    -----
            A -> Z                     positive_unate
            B -> Z                     positive_unate
```

ptc_shell> **show_lib_arc -from mylib/AN2/A**

```
from_lib_pin    to_lib_pin                    sense
------------    ----------                    -----
            A -> Z                     positive_unate
```

SEE ALSO

```
collections(2)
filter_collection(2)
foreach_in_collection(2)
get_attribute(2)
get_timing_arcs(2)
```

get_libs

Creates a collection of libraries loaded into constraint consistency. You can assign these libraries to a variable or pass them into another command.

SYNTAX

```
collection get_libs [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns | -of_objects objects
```

Data Types

```
   expression     string
   objects        list
   patterns       list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any libraries that match *patterns* (or *objects*), the expression is evaluated based on the library's attributes. If the expression evaluates to true, the library 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, modify the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. **-regexp** and **-exact** 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. **-exact** and **-regexp** are mutually exclusive.

-of_objects ***objects***

Creates a collection of libraries that contain the specified objects. In this case, each object is either a named library cell or a library cell collection. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both.

patterns

Matches library 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 lib. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_libs** command creates a collection of libraries from those currently loaded into constraint consistency that match certain criteria. The command returns a collection if any libraries 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*. 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_libs** 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_libs** result to a variable.

When issued from the command prompt, **get_libs** 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_libs** 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_libs** 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 all loaded libraries. Although the output looks like a list, it is just a display. Note that a complete listing of libraries is available using the **list_libraries** command.

```
ptc_shell> get_libs *
{"misc_cmos", "misc_cmos_io"}
```

The following example shows that given a library collection, you can remove those libraries. Note that you cannot remove libraries if they are referenced by a design.

```
ptc_shell> remove_lib [get_libs misc*]
Removing library misc_cmos...
Removing library misc_cmos_io...
```

SEE ALSO

```
collections(2)
filter_collection(2)
list_libraries(2)
query_objects(2)
regexp(2)
remove_lib(2)
collection_result_display_limit(3)
```

get_message_ids

Get application message ids

SYNTAX

```
string get_message_ids [-type severity]
[pattern]

string severity
string pattern
```

ARGUMENTS

-type *severity*

Filter ids based on type (Values: Info, Warning, Error, Severe, Fatal)

pattern

Get IDs matching pattern (default: *)

DESCRIPTION

The **get_message_ids** command retrieves the error, warning and informational messages used by the application. The result of this command is a Tcl formatted list of all message ids. Information about the id can be queried with the **get_message_info** command.

EXAMPLES

The following code finds all error messages and makes the application stop script execution when one of

these messages is encountered.

```
foreach id [get_message_ids -type Error] {
  set_message_info -stop_on -id $id
}
```

SEE ALSO

print_message_info(2)
set_message_info(2)
suppress_message(2)

get_message_info

Returns information about diagnostic messages.

SYNTAX

```
integer get_message_info
   [-error_count | -warning_count | -info_count
     | -limit l_id | -occurrences o_id | -suppressed s_id | -id i_id]
```

Data Types

```
   l_id      string
   o_id      string
   s_id      string
   i_id        string
```

ARGUMENTS

-error_count

Returns the number of error messages issued so far.

-warning_count

Returns the number of warning messages issued so far.

-info_count

Returns the number of informational messages issued so far.

-limit *l_id*

Returns the current user-specified limit for a given message ID. The limit was set with the **set_message_info** command.

-occurrences *o_id*

Returns the number of occurrences of a given message ID.

-suppressed *s_id*

Returns the number of times a message was suppressed either using **suppress_message** or due to

exceeding a user-specified limit.

-id *i_id*

Returns information about the specified message. The information is returned as a Tcl list compatible with the array set command.

DESCRIPTION

The **get_message_info** command retrieves information about error, warning, and informational messages. For example, if the following message is generated, information about it is recorded:

Error: unknown command 'wrong_command' (CMD-005)

It is useful to be able to retrieve recorded information about generated diagnostic messages. For example, you can stop a script after a certain number of errors have occurred, or monitor the number of messages issued by a single command.

You can also find out how many times a specific message has occurred, or how many times it has been suppressed. Also, you can find out if a limit has been set for a particular message ID.

EXAMPLES

The following example uses the **get_message_info** command to count the number of errors that occurred during execution of a specific command, and to return from the procedure if the error count exceeds a given amount:

```
prompt> proc \
  do_command {limit} {
    set current_errors [get_message_info -error_count]
    command
    set new_errors [get_message_info -error_count]
    if {[expr $new_errors - $current_errors] > $limit} {
      return -code error "Too many errors"
    }
    ...
  }
```

The following example uses the **get_message_info** command to retrieve information on the CMD-014 message:

```
prompt> get_message_info -id CMD-014
id CMD-014 severity Error limit 0 occurrences 0 suppressed 0 message
{Invalid %s value '%s' in list.}
```

SEE ALSO

```
print_message_info(2)
set_message_info(2)
suppress_message(2)
get_message_ids(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]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-top_net_of_hierarchical_group]
[-segments]
[-boundary_type btype]
   patterns | -of_objects objects
```

Data Types

expression	string
objects	list
patterns	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." You cannot use **-hierarchical** with **-of_objects**.

-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. **-regexp** and **-exact** 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. **-exact** and **-regexp** are mutually exclusive.

-top_net_of_hierarchical_group

Keep only the top net of a hierarchical group. When more than one hierarchical net of the same group is specified (local nets at various hierarchical levels of the same physical net), only the net closest to the top of the hierarchy is saved with the collection. In the case of multiple nets at the same level, the first net specified is kept. Although this option can be used when there are multiple nets specified, it is best used in combination with **-segments** and with a single net.

-segments

Returns all global segments for given nets. This modifies the initial search which matches *patterns* or *objects*. It is applied after filtering, but before the **-top_net_of_hierarchical_group** is determined. For each net, all global segments which are part of that net are added to the result collection. Global net segments are all those physically connected across all hierarchical boundaries. Although this option can be used with other options and when there are multiple nets specified, it is best used with a single net. When combined with the **-top_net_of_hierarchical_group** option, you can isolate the highest net segment of a physical net.

-boundary_type btype

Specifies what to do when getting nets of boundary pins. This option requires the **-of_objects** option. Allowed values are lower, upper, and both, meaning the net inside the hierarchical block, outside the hierarchical block, or both nets, respectively. The option has no meaning for non-hierarchical pins. Note that The "upper" value is less useful, as getting pins of a hierarchical net will return the pin outside the hierarchical block, and a subsequent get_nets would find the upper hierarchical net. Using upper on a hierarchical pin is the same as omitting the option.

-of_objects *objects*

Creates a collection of nets connected to the specified objects. Each object is either a named pin, a pin collection, a named cell or cell collection. **-of_objects** and *patterns* 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. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 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; 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_nets** 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_nets** result to a variable.

When issued from the command prompt, **get_nets** 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_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** 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.

```
ptc_shell> 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.

```
ptc_shell> current_instance block1
block1
ptc_shell> set pinsel [get_pins {o_reg1/QN o_reg2/QN}]
{"o_reg1/QN", "o_reg2/QN"}
ptc_shell> query_objects [get_nets -of_objects $pinsel]
{"NET1QNX", "NET2QNX"}
```

This example shows how to use the **-top_net_of_hierarchical_group** and **-boundary_type** options. Given the following circuit:

```
  u1                                  u2
-----------------+              +-------------------
                 |              |
 u1/Z -- net5 ---+--- topnet ---+--- net7 -- u1/A
                 |              |
-----------------+              +-------------------
```

there are hierarchical cells u1 and u2 at this level, connected by a net *topnet*. Within u1 is a pin u1/Z driving *net5*, and withing u2 is a pin u1/A being driven by *net7*. These three nets are physically the same net. Assume these are the only nets in the design. Notice the difference in the results of the following two **get_nets** commands:

```
ptc_shell> get_nets * -hierarchical
{"topnet", "u1/net5", "u2/net7"}
ptc_shell> get_nets * -hierarchical -top_net_of_hierarchical_group
{"topnet"}
```

Assume that at the top level, topnet is connected to pins u2/in and u1/out. Here are some examples of **-boundary_type**. Notice now in this case, the "upper" type does not add value.

```
ptc_shell> get_nets -of_objects u2/in
{"topnet"}
ptc_shell> get_nets -of_objects u2/in -boundary_type upper
{"topnet"}
ptc_shell> get_nets -of_objects u2/in -boundary_type lower
{"u2/net7"}
ptc_shell> get_nets -of_objects u2/in -boundary_type both
{"u2/net7", "topnet"}
ptc_shell> get_nets -boundary lower -of_objects \
             [get_pins -of_objects [get_nets topnet]]
{"u1/net5", "u2/net7"}
```

SEE ALSO

collections(2)
filter_collection(2)
get_pins(2)
link_design(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)

get_object_name

Gets the name of objects in the given collection.

SYNTAX

```
string get_object_name
   collection
```

Data Types

```
   collection       string
```

ARGUMENTS

collection

Specifies the collection.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_object_name** command is a convenient way to get the *full_name* attribute of objects. When multiple objects are passed to the **get_attribute** command, the values are returned as a Tcl list, containing one entry for each object.

EXAMPLES

The following example shows how to use the **get_object_name** command:

```
ptc_shell> get_object_name [get_cells i1]
i1
ptc_shell> get_attribute [get_cells {i1 i2}] full_name
{i2 i3} i1
ptc_shell> get_object_name [get_cells i*]
{i1 i2 i3}
```

SEE ALSO

collections(2)
get_attribute(2)

get_output_delays

Creates a collection of output_delays in the current scenario. You can assign these output_delays to a variable or pass them into another command.

SYNTAX

```
collection get_output_delays
   [-quiet]
   [-filter expression]
   -of_objects objects
```

Data Types

```
   expression    string
   objects       list
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with *expression*. For any output_delays on *objects*, the expression is evaluated based on the output_delay's attributes. If the expression evaluates to true, the output_delay is included in the result.

-of_objects *objects*

Creates a collection of output_delays defined on the specified objects. Each object is either a named pin, a pin collection, a named port or port collection.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_output_delays** command creates a collection of output_delays matching *patterns* in the current scenario and which pass the filter (if specified). If no objects match the criteria, the empty string is returned.

EXAMPLES

This example sets variable inDel to the collection of output_delays defined on ports IN*.

```
ptc_shell> set inDel [get_output_delays -of [get_ports IN*]]
```

SEE ALSO

collections(2)
set_output_delay(2)

get_output_unit

Returns a string representing the output unit for one unit type.

SYNTAX

```
string get_output_unit
   -type unit_type
   [-numeric]
```

Data Types

```
   unit_type     string
```

ARGUMENTS

-type *unit_type*

Specifies the type of quantity, which must be one of the values: "time", "resistance", "capacitance", "voltage", "current", or "power".

-numeric

If specified, return the value as a number representing the unit value in terms of the base unit for that unit type. Base units are Seconds, Ohms, Farads, Volts, Amperes, and Watts.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command returns a string representing the output unit for one unit type. The tool maintains separate units for input and output. There are output unit values for time, resistance, capacitance, voltage, current, and power.

EXAMPLES

The following example shows how to get the output unit for time. With the **-numeric** option, the value is returned in terms of the base unit for that quantity type (seconds, for time).

```
ptc_shell> get_output_unit -type time
1.00ps
ptc_shell> get_output_unit -type time -numeric
1e-12
```

SEE ALSO

set_output_units(2)

get_path_pins

Performs an analysis of all the paths satisfying the specification.

SYNTAX

```
collection get_path_pins
   [-rise | -fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-quiet]
```

Data Types

```
   from_list             list
   rise_from_list        list
   fall_from_list        list
   through_list          list
   rise_through_list     list
   fall_through_list     list
   to_list               list
   rise_to_list          list
   fall_to_list          list
```

ARGUMENTS

-rise

Indicates that only rising paths are to be analyzed. If neither **-rise** nor **-fall** is specified, both rising and falling delays are analyzed. Rise refers to a rising value at the path endpoint.

-fall

Indicates that only falling path delays are to be analyzed. If neither **-rise** nor **-fall** is specified, both rising and falling delays are analyzed. Fall refers to a falling value at the path endpoint.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, cells and nets through which the multicycle paths must pass. Nets are interpreted to imply the leaf-level driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than one time in a single command invocation.

-rise_through ***rise_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects. You can specify **-rise_through** more than one time in a single command invocation.

-fall_through ***fall_through_list***

This option is similar to the **-through** option, but applies only to paths with a fall transition at the specified objects. You can specify **-fall_through** more than one time in a single command invocation.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can

use only one of **-to**, **-rise_to**, and **-fall_to**.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command creates a collection of pins for all of paths satisfying the specification given.

SEE ALSO

```
set_false_path(2)
set_multicycle_path(2)
```

get_path_sets

Returns all of the path sets satisfying the specification.

SYNTAX

```
Boolean get_path_sets
   [-rise | -fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-path_type format]
   [-max_endpoints]
   [-traverse_disabled]
   [-unconstrained]
   [-valid]
```

Data Types

```
   from_list          list
   rise_from_list     list
   fall_from_list     list
   through_list       list
   rise_through_list  list
   fall_through_list  list
   to_list            list
   rise_to_list       list
   fall_to_list       list
   format             string
```

ARGUMENTS

-rise

Indicates that only rising paths are to be analyzed. If neither **-rise** nor **-fall** is specified, both rising and falling delays are analyzed. Rise refers to a rising value at the path endpoint.

-fall

Indicates that only falling path delays are to be analyzed. If neither **-rise** nor **-fall** is specified, both rising and falling delays are analyzed. Fall refers to a falling value at the path endpoint.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, cells and nets through which the multicycle paths must pass. Nets are interpreted to imply the net segment's local driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than one time in a single command invocation.

-rise_through ***rise_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects. You can specify **-rise_through** more than one time in a single command invocation.

-fall_through ***fall_through_list***

This option is similar to the **-through** option, but applies only to paths with a fall transition at the specified objects. You can specify **-fall_through** more than one time in a single command invocation.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-path_type ***format***

The format might be "end", "summary". The "end" format reports the paths for each endpoint specified. The "summary" format report the paths between each pair of start and end points.

-max_endpoints

This option reduces the paths displayed to at most the given number of outputs.

-traverse_disabled

This option causes the returned **path_set** collection to include paths that are disabled due to constraints such as case, **set_disable_timing**, and **set_max_delay**.

-unconstrained

This option causes the returned **path_set** collection to include paths that are not normally legal timing paths. These are paths the are **-from** a pin that does not launch timing paths, or paths that are -to a pin that is not constrained. This option is helpful in debugging situations where the **all_fanin** or **all_fanout** command returns too much information.

-valid

This option causes the returned **path_set** collection to not include paths that are false or having clock group exceptions on them.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_path_sets** command returns as a Tcl collection of **path_set**, the paths satisfying the given specification. The analysis performed here to collect the paths is identical to the one performed by the **analyze_paths** command.

EXAMPLES

The following example shows how to use the command:

```
ptc_shell> set path_set_A [get_path_set -from [get_pins A]]
ptc_shell> set exceptions_A [get_attribute $path_set_A dominant_exceptions]
```

```
The following command lists all the attributes available on the "path_set" object:

ptc_shell> list_attributes -application -class path_set
****************************************
Report : List of Attribute Definitions
Version: ...
Date   : ...
****************************************

Properties:
    A - Application-defined
    U - User-defined
    I - Importable from design/library (for user-defined)

Attribute Name            Object     Type       Properties  Constraints
------------------------------------------------------------------------------
disabled                  path_set   boolean    A
dominant_exceptions       path_set   collection A
endpoint                  path_set   collection A
endpoint_clock            path_set   collection A
endpoint_clock_is_inverted
                          path_set   boolean    A
endpoint_clock_open_edge_type
                          path_set   string     A
max_fall_count            path_set   int        A
max_rise_count            path_set   int        A
min_fall_count            path_set   int        A
min_rise_count            path_set   int        A
object_class              path_set   string     A
overriden_exceptions      path_set   collection A
startpoint                path_set   collection A
startpoint_clock          path_set   collection A
startpoint_clock_is_inverted
                          path_set   boolean    A
startpoint_clock_open_edge_type
                          path_set   string     A
```

SEE ALSO

analyze_paths(2)
get_path_pins(2)

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]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-leaf]
   patterns | -of_objects objects
```

Data Types

```
   expression       string
   patterns         list
   objects          list
```

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.

-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. **-regexp** and **-exact** 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. **-exact** and **-regexp** 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 are included in the collection. In addition, hierarchical boundaries are crossed to find pins on leaf cells.

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. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

-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. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both. In addition, you cannot use **-hierarchical** with **-of_objects**.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 the *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. 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 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 indecies 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 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; 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_pins** 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_pins** result to a variable.

When issued from the command prompt, **get_pins** 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_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** 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.

```
ptc_shell> 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.

```
ptc_shell> set csel [get_cells o_reg1]
{"o_reg1"}
ptc_shell> 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.

```
ptc_shell> get_pins -of_objects [get_nets NET1]
{"i2/a", "reg1/QN"}
ptc_shell> 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.

```
ptc_shell> 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_ports

Creates a collection of ports from the current design or instance. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection get_ports
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [patterns | -of_objects objects]
```

Data Types

```
   expression      string
   patterns        list
   objects         list
```

ARGUMENTS

-filter *expression*

Filters the collection with the *expression* option. 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.

-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

When combined with the *-regexp* option, makes matches case-insensitive. You can use the *-nocase* option only when you also use the *-regexp* option.

-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.

-of_objects *objects*

Creates a collection of ports connected to the specified objects. Each object is a named net collection. The *-of_objects* and *patterns* options are mutually exclusive; you can specify only one.

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* and *-of_objects* options are mutually exclusive; you can specify only one.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_ports** command creates a collection of ports in the current design or instance that match certain criteria. The command returns a collection if any ports match the *patterns* option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

If the **port_search_in_current_instance** variable is true, then the **get_ports** command gets the ports of current instance. If the **port_search_in_current_instance** variable is false (the default), then the **get_ports** command gets the ports of the current design.

Regular expression matching is the same as in the **regexp** Tcl command. When using the *-regexp* option, take care in the way you quote the *patterns* and filter the *expression* option; 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 simply by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_ports** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_ports** command result to a variable.

When issued from the command prompt, the **get_ports** command behaves as though the **query_objects** command 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 **collection_result_display_limit** variable.

The "implicit query" property of the **get_ports** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_ports** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page. In addition, refer to the **all_inputs** and **all_outputs** man pages, 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.

```
ptc_shell> 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.

```
ptc_shell> set_driving_cell -cell FD2 -library my_lib [get_ports in*]
```

The following example reports ports connected to nets matching the pattern "bidir*".

```
ptc_shell> 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)
port_search_in_current_instance(3)
collection_result_display_limit(3)

get_rule_property

Gets a property value on a rule. Properties affect how a specific rule check is performed, or how the output of a rule violation is presented.

SYNTAX

```
string get_rule_property
   property_name
   rule
```

Data Types

```
   property_name     string
   rule              string
```

ARGUMENTS

property_name

The name of a valid property on the specified rule.

rule

Specifies the name of an existing rule.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Obtains a property value for one rule. Properties can control rule checking; for example, the limit of some comparison. Properties can also control the output; for example, the maximum number of pins shown in the more info section of a violation.

EXAMPLES

The following example gets the **max_percent** property for rule **EXD_0009** and assigns it to a variable.

```
ptc_shell> set maxPercentVar [get_rule_property max_percent EXD_0009]
```

SEE ALSO

analyze_design(2)
create_rule(2)
set_rule_property(2)

get_rule_violations

Creates a collection of rule violations. You can assign the resulting collection to a variable or pass it into another command.

SYNTAX

```
collection get_rule_violations
   -of_objects objects
   [-filter expression]
   [-block_to_top_cells cell_list]
   [-quiet]
```

Data Types

```
   objects          list
   expression       string
   cell_list        list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any rules that match *patterns* (or *objects*) the expression is evaluated based on the rule's attributes. If the expression evaluates to true, the rule is included in the result.

-of_objects *objects*

Creates a collection of rules contained in the specified list. In this case, each entry is either a named rule or a collection of rules.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-block_to_top_cells *cell_list*

Get rule violations for only the specific set of instances in block-to-top rules.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_rule_violations** command creates a collection of rule violations defined in the session that match certain criteria. The command returns a collection if any rule violations match the *objects* argument and pass the filter (if specified). If no rules match the criteria, the empty string is returned.

EXAMPLES

The following example gets a collection of rule violations for rule CLK_0020.

```
ptc_shell> get_rule_violation -of [get_rules CLK_0020]
```

SEE ALSO

```
analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
enable_rule(2)
get_rulesets(2)
```

get_rules

Creates a collection of rules. You can assign the resulting collection to a variable or pass it into another command.

SYNTAX

```
collection get_rules
   [-filter expression]
   [-quiet]
   patterns | -of_objects objects
```

Data Types

```
   expression       string
   patterns         list
   objects          list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any rules that match *patterns* (or *objects*) the expression is evaluated based on the rule's attributes. If the expression evaluates to true, the rule is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-of_objects *objects*

Creates a collection of rules contained in the specified rulesets. In this case, each entry is either a named ruleset or a collection of rulesets. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both.

patterns

Matches rule names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type rule. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_rules** command creates a collection of rules defined in the session that match certain criteria. The command returns a collection if any rules match the *patterns* or *objects* and pass the filter (if specified). If no rules match the criteria, the empty string is returned.

EXAMPLES

The following example performs the **disable_rule** command on the collection of rules matching pattern "CSL_*".

```
ptc_shell> disable_rule [get_rules CSL_*]
```

SEE ALSO

analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
enable_rule(2)
get_rulesets(2)

get_rulesets

Creates a collection of rulesets. You can assign the resulting collection to a variable or pass it into another command.

SYNTAX

```
collection get_rulesets
   [-filter expression]
   [-quiet]
   patterns | -of_objects objects
```

Data Types

```
   expression     string
   patterns       list
   objects        list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any rulesets that match *patterns* (or *objects*) the expression is evaluated based on the ruleset's attributes. If the expression evaluates to true, the ruleset is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-of_objects *objects*

Creates a collection of rulesets containing the specified rules. In this case, each entry is either a named rule or a collection of rules. **-of_objects** and *patterns* are mutually exclusive; you must specify one, but not both.

patterns

Matches ruleset names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type ruleset. *patterns* and **-of_objects** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_rulesets** command creates a collection of rulesets defined in the session that match certain criteria. The command returns a collection if any rulesets match the *patterns* or *objects* and pass the filter (if specified). If no rulesets match the criteria, the empty string is returned.

EXAMPLES

The following example performs the **disable_rule** command on the rules in ruleset "my_set1"

```
ptc_shell> disable_rule [get_rulesets my_set1]
```

SEE ALSO

```
analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
get_rules(2)
```

get_scenarios

Creates a collection of scenarios in the current design. You can assign these scenarios to a variable or pass them into another command. The get_mode command is a synonym for the get_scenario command.

SYNTAX

```
collection get_scenarios
   [-quiet]
   [-regexp]
   [-nocase]
   [-filter expression]
   patterns
```

Data Types

```
   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 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.

-nocase

When combined with **-regexp**, makes matches case-insensitive. You can use **-nocase** only when you also use **-regexp**.

-filter *expression*

Filters the collection with *expression*. For any scenarios that match *patterns*, the expression is evaluated based on the scenario's attributes. If the expression evaluates to true, the scenario is included in the result.

patterns

Matches scenario names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_scenarios** command creates a collection of scenarios matching *patterns* in the current design and which pass the filter (if specified). If no objects match the criteria, the empty string is returned.

EXAMPLES

This example passes a collection of scenarios with names starting with "Test" to the analyze_design command.

```
ptc_shell> analyze_design -scenarios [get_scenarios Test*] -output_dir .
```

SEE ALSO

all_scenarios(2)
collections(2)
create_scenario(2)

get_timing_arcs

Creates a collection of timing arcs for custom reporting and other processing. You can assign these timing arcs to a variable and get the needed attribute for further processing.

SYNTAX

```
string get_timing_arcs
   [-to to_list]
   [-from from_list]
   [-of_objects object_list]
   [-filter expression]
   [-quiet]
```

Data Types

```
   to_list            list
   from_list          list
   object_list        list
   expression         string
```

ARGUMENTS

-to *to_list*

Specifies a list of to pins or to ports to get timing arcs for. All backward arcs from the specified pins or ports are considered.

-from *from_list*

Specifies a list of from pins or from ports to get timing arcs for. All forward arcs from the specified pins or ports are considered.

-of_objects *object_list*

Specifies cells or nets to get timing arcs for. If a cell is specified, all cell_arcs of that cell are considered. If a net is specified, all net_arcs of that net are considered.

-filter *filter_str*

Specifies a string that comprises a series of logical expressions describing a set of constraints you want to place on the collection of arcs. Each subexpression of a filter expression is a relation contrasting an

attribute name with a value by means of an operator.

-quiet

Specifies that all messages are to be suppressed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **get_timing_arcs** command creates a collection of arcs for custom reporting or other operations. Use the **foreach_in_collection** command to iterate among the arcs in the collection. You can use the **get_attribute** command to obtain information about the arcs. You can also use the filter expression to filter the arcs that satisfy the specified conditions. The following attributes are supported on timing arcs:

```
delay_max_fall
delay_max_rise
delay_min_fall
delay_min_rise
from_pin
is_annotated_fall_max
is_annotated_fall_min
is_annotated_rise_max
is_annotated_rise_min
is_cellarc
is_disabled
is_user_disabled
mode
object_class
sdf_cond
sense
to_pin
```

One attribute of a timing arc is from_pin which is the pin or port from which the timing arc begins. In the same way, the to_pin is the pin or port at which the timing arc ends. In order to get more information about the from_pin and the to_pin, use the **get_attributes** command. The **object_class** attribute holds the name of the timing arc class **timing_arc**.

EXAMPLES

The following procedure gets the same arguments as the **get_timing_arcs** command and prints out some of the attributes of the timing arcs.

```
proc format_float {number {format_str "%.2f"}} {

  switch -exact -- $number {
    UNINIT { }
    INFINITY { }
    default {set number [format $format_str $number]}
```

```
  }
  return $number;
}

proc show_arcs {args} {

  set arcs [eval [concat get_timing_arcs $args]]
  echo [format "%15s    %-15s  %8s %8s  %s" "from_pin" "to_pin" \
           "rise" "fall" "is_cellarc"]
  echo [format "%15s    %-15s  %8s %8s  %s" "--------" "------" \
           "----" "----" "----------"]

  foreach_in_collection arc $arcs {
    set is_cellarc [get_attribute $arc is_cellarc]
    set fpin [get_attribute $arc from_pin]
    set tpin [get_attribute $arc to_pin]
    set rise [get_attribute $arc delay_max_rise]
    set fall [get_attribute $arc delay_max_fall]

    set from_pin_name [get_attribute $fpin full_name]

    set to_pin_name [get_attribute $tpin full_name]
    echo [format "%15s -> %-15s  %8s %8s  %s" \
        $from_pin_name $to_pin_name \
        [format_float $rise] [format_float $fall] \
        $is_cellarc]
  }
}
```

```
ptc_shell> show_arcs -of_objects ffa

       from_pin    to_pin              rise     fall  is_cellarc
       --------    ------              ----     ----  ----------
         ffa/CP -> ffa/D               0.85     0.85  true
         ffa/CP -> ffa/D               0.40     0.40  true
         ffa/CP -> ffa/Q               1.34     1.42  true
         ffa/CP -> ffa/QN              2.38     1.98  true
         ffa/CD -> ffa/Q               1.00     0.82  true
         ffa/CD -> ffa/QN              1.78     1.00  true


ptc_shell> show_arcs -from ffa/CP

       from_pin    to_pin              rise     fall  is_cellarc
       --------    ------              ----     ----  ----------
         ffa/CP -> ffa/D               0.85     0.85  true
         ffa/CP -> ffa/D               0.40     0.40  true
         ffa/CP -> ffa/Q               1.34     1.42  true
         ffa/CP -> ffa/QN              2.38     1.98  true


ptc_shell> show_arcs -from ffa/Q

       from_pin    to_pin              rise     fall  is_cellarc
       --------    ------              ----     ----  ----------
          ffa/Q -> QA/A                0.00     0.00  false


ptc_shell> show_arcs -to ffa/*

       from_pin    to_pin              rise     fall  is_cellarc
       --------    ------              ----     ----  ----------
         ffa/CP -> ffa/D               0.85     0.85  true
         ffa/CP -> ffa/D               0.40     0.40  true
```

```
   a/Z -> ffa/D              0.00     0.00  false
   CLK -> ffa/CP             0.00     0.00  false
 RESET -> ffa/CD             0.00     0.00  false
ffa/CP -> ffa/Q              1.34     1.42  true
ffa/CD -> ffa/Q              1.00     0.82  true
ffa/CP -> ffa/QN             2.38     1.98  true
ffa/CD -> ffa/QN             1.78     1.00  true
```

SEE ALSO

collections(2)
foreach_in_collection(2)
get_attribute(2)

get_violation_info

Gets the values of parameters or violation details attributes of rule violation instance.

SYNTAX

```
collection get_violation_info
   [-parameter parameter_name]
   [-attribute violation_detail_name]
   objects
```

Data Types

```
   parameter_name              string
   violation_detail_name       string
   objects                   list
```

ARGUMENTS

-parameter *parameter_name*

Queries the rule violation parameter value associated with *parameter_name*.

-attribute *violation_detail_name*

Queries the rule violation detail value associated with *violation_detail_name*.

objects

objects is a collection of rule violation objects.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command gets the values of parameters or violation details attributes of rule violation instances. It creates either a heterogeneous collection of objects or a list of strings.

To obtain the list of available parameters, query the parameters attribute on the corresponding rule. To obtain the list of violation_detail_name available for a rule, query the "violation_details" attribute on the corresponding rule.

This command allows only one parameter or violation detail attribute to be queried at one time. In addition, the violation objects passed to the command should always be of the same rule type.

EXAMPLES

The following example gets a rule violation for the CLK_0033 rule; it then gets the clock parameter of that violation.

```
ptc_shell> set rule_viol [index_collection [get_rule_violations -of_obejcts CLK_0033] 0]
ptc_shell> get_violation_info -parameter clock $rule_viol
{"CLK2"}
```

The following example shows how to query available violation details for a given rule using report_attribute and use that information to further query the clock list of a corresponding rule violation. Then it retrieves the waveform attribute of the clock list.

```
ptc_shell> report_attribute -application [get_rule CGR_0001]
****************************************
Report : Attribute
Design : CGR_0001
****************************************

Design          Object          Type        Attributes          Value
--------------------------------------------------------------------------------
CGR_0001        CGR_0001        string      description         Clocks which are generated from
                                                                the same master clock cannot be
                                                                asynchronous with each other.
CGR_0001        CGR_0001        string      full_name           CGR_0001
CGR_0001        CGR_0001        boolean     is_enabled          true
CGR_0001        CGR_0001        string      message             {Clocks generated from the same
                                                                master clock } { are declared
                                                                as asynchronous.}
CGR_0001        CGR_0001        string      object_class        rule
CGR_0001        CGR_0001        string      parameters          {master_clock}
CGR_0001        CGR_0001        string      severity            error
CGR_0001        CGR_0001        string      violation_details   {clock1_list} {clock2_list}
1

ptc_shell> set clock_list [get_violation_info -attribute clock1_list \
[get_rule_violation -of_objects CGR_0001]]
{"clk2"}
ptc_shell> get_attribute $clock_list waveform

{0.000000 4.000000}
```

SEE ALSO

```
analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
enable_rule(2)
get_attribute(2)
get_rulesets(2)
get_rule_violations(2)
report_attribute(2)
```

getenv

Returns the value of a system environment variable.

SYNTAX

```
string getenv
   variable_name
```

Data Types

```
   variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of the environment variable to be retrieved.

DESCRIPTION

The **getenv** command searches the system environment for the specified *variable_name* and sets the result of the command to the value of the environment variable. If the variable is not defined in the environment, the command returns a Tcl error. The command is catchable.

Environment variables are stored in the **env** Tcl array variable. The **getenv**, **setenv**, and **printenv** environment commands are convenience functions to interact with this array.

The application you are running inherited the initial values for environment variables from its parent process (that is, the shell from which you invoked the application). If you set the variable to a new value using the **setenv** command, you see the new value within the application and within any new child processes you initiate from the application using the **exec** command. However, these new values are not exported to the parent process. Further, if you set an environment variable using the appropriate system command in a shell you invoke using the **exec** command, that value is not reflected in the current application.

See the **set**, **unset**, and **printvar** commands for information about working with non-environment

variables.

EXAMPLES

In the following example, **getenv** returns you to your home directory:

```
prompt> set home [getenv "HOME"]
/users/disk1/bill

prompt> cd $home

prompt> pwd
/users/disk1/bill
```

In the following example, **setenv** changes the value of an environment variable:

```
prompt> getenv PRINTER
laser1

prompt> setenv PRINTER "laser3"
laser3

prompt> getenv PRINTER
laser3
```

In the following example, the requested environment variable is not defined. The error message shows that the Tcl variable **env** was indexed with the value UNDEFINED, which resulted in an error. In the second command, **catch** is used to suppress the message.

```
prompt> getenv "UNDEFINED"
Error: can't read "env(UNDEFINED)": no such element in array
        Use error_info for more info. (CMD-013)

prompt> if {[catch {getenv "UNDEFINED"} msg]} {
          setenv UNDEFINED 1
        }
```

SEE ALSO

```
catch(2)
exec(2)
printenv(2)
printvar(2)
set(2)
setenv(2)
unsetenv(2)
unset(2)
```

group_path

Groups paths for cost function calculations supported for compatibility with Galaxy tools.

SYNTAX

```
Boolean group_path
   -name group_name
   -default
   [-weight weight_value]
   [-from from_list]
   [-rise_from rise_from_list]
   [-fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list]
   [-rise_to rise_to_list]
   [-fall_to fall_to_list]
   [-comment comment_string]
```

Data Types

```
   group_name             string
   weight_value           float
   from_list              list
   rise_from_list         list
   fall_from_list         list
   through_list           list
   rise_through_list      list
   fall_through_list      list
   to_list                list
   rise_to_list           list
   fall_to_list           list
   comment_string         string
```

ARGUMENTS

-name *group_name*

Specifies a name for the group. If a group with this name already exists, constraint consistency adds the paths or endpoints to that group. If a group with this name does not exist, constraint consistency creates a new group. You must specify the *-name* option unless you use the *-default* option; the *-name*

and *-default* options are mutually exclusive.

-default

Specifies that endpoints or paths are moved to the default group and removed from the current group. You must specify the *-default* option unless you use the *-name* option; the *-default* and *-name* options are mutually exclusive.

-weight *weight_value*

Specifies a cost function weight for this group.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of the *-from*, *-rise_from*, and *-fall_from* options.

-rise_from *rise_from_list*

Same as the *-from* option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by the rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the *-from*, *-rise_from*, and *-fall_from* options.

-fall_from *fall_from_list*

Same as the *-from* option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by the falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the *-from*, *-rise_from*, and *-fall_from* options.

-through *through_list*

Specifies a list of path throughpoints (ports, pins, cells, or nets).

-rise_through *rise_through_list*

Similar to the *-through* option, but applies only to paths with a rising transition at the specified objects.

-fall_through *fall_through_list*

Similar to the *-through* option, but applies only to paths with a falling transition at the specified objects.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If you specify a cell, one path endpoint on that cell is affected.

-rise_to *rise_to_list*

Same as the *-to* option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the *-to*, *-rise_to*, and *-fall_to* options.

-fall_to *fall_to_list*

Same as the *-to* option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the *-to*, *-rise_to*, and *-fall_to* options.

-comment *comment_string*

Associates a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Groups a set of paths or endpoints for cost function calculations in optimization and analysis. Since constraint consistency does not perform any costing, this command is just supported along with report_path_group just for compatibility with Galaxy tools.

To undo the **group_path** command, use the **remove_path_group** command. To report path group information for a design, use the **report_path_group** command.

EXAMPLES

The following example groups all endpoints clocked by CLK1A or CLK1B into a new group 'group1' with weight = 2.0.

```
ptc_shell> group_path -name group1 \
-weight 2.0 -to {CLK1A CLK1B}
```

SEE ALSO

create_clock(2)
current_design(2)
remove_path_group(3)
report_path_group(2)
reset_design(2)
set_input_delay(2)
set_output_delay(2)

gui_start

Starts the application GUI.

SYNTAX

```
string gui_start

   [-file script]
   [-no_windows]
   [-- x_args ...]

   string script
```

ARGUMENTS

-file *script*

The given script file is sourced before the GUI starts.

-no_windows

The GUI starts without showing the default window.

-- x_args ...

X11-specific arguments to pass to the X connection.

DESCRIPTION

This command starts the application GUI from the shell prompt. It is ignored if the application GUI has already been started. start_gui is an alias for gui_start.

Note the application can only ever connect to one X server during program execution, so changing the value of the DISPLAY environment variable after the first successful gui_start command has no effect.

EXAMPLES

The following example starts the application GUI.

shell> **gui_start**

SEE ALSO

gui_stop(2)
start_gui(2)
stop_gui(2)

gui_stop

Stops the application GUI.

SYNTAX

```
string gui_stop
```

ARGUMENTS

None.

DESCRIPTION

This command stops the application GUI and returns to the shell prompt. It is ignored if the application GUI has not been started or has been stopped. stop_gui is an alias for gui_stop.

EXAMPLES

The following example stops the application GUI and returns to the shell prompt.

shell> **gui_stop**

SEE ALSO

gui_start(2)

```
start_gui(2)
stop_gui(2)
```

gui_write_window_image

Saves an image of a view window or toplevel window in the specified image file

SYNTAX

```
status gui_write_window_image
   -file filename
   [ -format image_format ]
   [ -window window_name ]
   [ -size window_size ]
```

Data Types

```
   filename      string
   image_format  string
   window_name   string
   window_size   {width height}
```

ARGUMENTS

-file *filename*

Specifies the name of the file in which the tool saves the window image.

If you include a file name extension, it determines the image format unless you also specify the *-format* option.

-format *png* | *xpm* | *jpg* | *bmp*

Specifies the image format to be used in the file. The default value is png.

Note that if the format that you specify with the *-format* option is different from the format specified by the file name extension, the *-format* value takes precedence and the tool appends an additional extension to the file name.

-window *window_name*

Specifies the name of the window for which you want to save an image in the specified image file.

You can view a list of the window names for all the open toplevel and view windows by using the *gui_get_window_ids* command.

By default, the tool saves an image of the most recently active view window.

-size *window_size*

The size of the image to be generated. If not specified then the full window size is used.

DESCRIPTION

The **gui_write_window_image** takes a snapshot of a specified toplevel window or child view window in a specified image format and writes the result to a specified file.

EXAMPLES

The following example writes a snapshot of the TopLevel.1 window to a image file "snapshot.png".

```
shell> gui_write_window_image -window TopLevel.1 -file snapshot.png
```

The following example writes a snapshot of the Layout.1 view to a image file "layout.png" of size 800 pixels by 600 pixels.

```
shell> gui_write_window_image -window Layout.1 -file layout.png -size {800 600}
```

SEE ALSO

```
gui_get_window_ids(2)
gui_get_current_window(2)
```

help

Displays quick help for one or more commands.

SYNTAX

```
string help
   [-verbose]
   [-groups]
   [pattern]
```

Data Types

```
   pattern      string
```

ARGUMENTS

-verbose

Displays the command options; for example *command_name* **-help**.

-groups

Displays a list of command groups only.

pattern

Displays commands matching the specified pattern.

DESCRIPTION

The **help** command is used to get quick help for one or more commands or procedures. This is not the same as the **man** command that displays reference manual pages for a command. There are many levels of help.

By typing the **help** command, a brief informational message is printed followed by the available command groups.

List all of the commands in a group by typing the group name as the argument to **help**. Each command is followed by a one-line description of the command.

To get a one-line help description for a single command, type **help** followed by the command name. You can specify a wildcard pattern for the name; for example, all commands containing the string "alias". Use the **-verbose** option to get syntax help for one or more commands. Use the **-groups** option to show only the command groups in the application. This option cannot be combined with any other option.

EXAMPLES

The following example lists the commands by command group:

```
prompt> help
Specify a command name or wild card pattern to get help on individual
commands. Use the '-verbose' option to get detailed option help for a
command. Commands also provide help when the '-help' option is passed to
the command.'

You can also specify a command group to get the commands available in
that group. Available groups are:

  Procedures:               Miscellaneous procedures
  Help:                     Help commands
  Builtins:                 Generic Tcl commands
 ...
```

The following example displays the list of procedures in the Procedures group:

```
prompt> help procedures
 ls                 # List files
 sh                 # Execute a shell command
```

The following example uses a wildcard character to display a one-line description of all commands beginning with *a*:

```
prompt> help a*
 alias              # Create a command which expands to words.
 append             # Builtin
 array              # Builtin
```

This example displays option information for the **source** command:

```
prompt> help -verbose source
 source             # Read a file and execute it as a script
   [-echo]              (Echo all commands)
   [-verbose]           (Display intermediate results)
   file_name            (Script file to read)
```

SEE ALSO

```
man(2)
sh_help_shows_group_overview(3)
```

help_attributes

Display help for attributes and object types

SYNTAX

```
string help_attributes [-verbose]
   [-application]
   [-user]
   [class_name]
   [attr_pattern]

string  class_name
string  attr_pattern
```

ARGUMENTS

-verbose

Display attribute properties.

-application

Only display application defined attributes.

-user

Only display user defined attributes.

class_name

Object class to retrieve help for.

attr_pattern

Show attributes matching pattern.

DESCRIPTION

The **help_attributes** command is used to get quick help for the set of available attributes. When this command is run with no arguments a brief informational message is printed followed by the available object classes.

EXAMPLES

```
prompt> help
cci_test> help_attributes
Specify an object type and an optional wild card pattern to get information
on the available attributes defined for the specified object type. Use the
-verbose option to get detailed information on the attributes

The available object types are:

  cell        net       pin
  ...
```

SEE ALSO

```
get_attribute(2)
help(2)
get_defined_attributes(2)
```

history

Displays or modifies the commands recorded in the history list.

SYNTAX

```
string history
   [-h]
   [-r]
   [argument_list]
```

Data Types

```
   argument_list      list
```

ARGUMENTS

-h

Displays the history list without the leading numbers. You can use this for creating scripts from existing history. You can then source the script with the **source** command. You can combine this option with only a single numeric argument. Note that this option is a nonstandard extension to Tcl.

-r

Reverses the order of output so that most recent history entries display first rather than the oldest entries first. You can combine this option with only a single numeric argument. Note that this option is a nonstandard extension to Tcl.

argument_list

Additional arguments to **history** (see DESCRIPTION).

DESCRIPTION

The **history** command performs one of several operations related to recently-executed commands

recorded in a history list. Each of these recorded commands is referred to as an "event." The most commonly used forms of the command are described below. You can combine each with either the **-h** or **-r** option, but not both.

- With no arguments, the **history** command returns a formatted string (intended for you to read) giving the event number and contents for each of the events in the history list.
- If a single, integer argument *count* is specified, only the most recent *count* events are returned. Note that this option is a nonstandard extension to Tcl.
- Initially, 20 events are retained in the history list. You can change the length of the history list using the following:

  history keep *count*

Tcl supports many additional forms of the **history** command. See the "Advanced Tcl History" section below.

EXAMPLES

The following examples show the basic forms of the **history** command. The first is an example of how to limit the number of events shown using a single numeric argument:

```
prompt> history 3
     7  set base_name "my_file"
     8  set fname [format "%s.db" $base_name]
     9  history 3
```

Using the **-r** option creates the history listing in reverse order:

```
prompt> history -r 3
     9  history -r 3
     8  set fname [format "%s.db" $base_name]
     7  set base_name "my_file"
```

Using the **-h** option removes the leading numbers from each history line:

```
prompt> history -h 3
set base_name "my_file"
set fname [format "%s.db" $base_name]
history -h 3
```

Advanced Tcl History

The **history** command performs one of several operations related to recently-executed commands recorded in a history list. Each of these recorded commands is referred to as an "event." When specifying an event to the **history** command, the following forms may be used:

- A number, which if positive, refers to the event with that number (all events are numbered starting at 1). If the number is negative, it selects an event relative to the current event; for example, **-1** refers to the previous event, **-2** to the one before that, and so on. Event **0** refers to the current event.

- A string selects the most recent event that matches the string. An event is considered to match the string either if the string is the same as the first characters of the event, or if the string matches the event in the sense of the **string match** command.

The **history** command can take any of the following forms:

history

Same as **history info**, described below.

history add *command [exec]*

Adds the *command* argument to the history list as a new event. If **exec** is specified (or abbreviated), the command is also executed and its result is returned. If **exec** is not specified, an empty string is returned.

history change *newValue [event_number]*

Replaces the value recorded for an event with *newValue*. The *event_number* specifies the event to replace, and defaults to the *current* event (not event **-1**). This command is intended for use in commands that implement new forms of history substitution and want to replace the current event (that invokes the substitution) with the command created through substitution. The return value is an empty string.

history clear

Erases the history list. The current keep limit is retained. The history event numbers are reset.

history event *[event_number]*

Returns the value of the event given by *event_number*. The default value of *event_number* is **-1**.

history info *[count]*

Returns a formatted string, giving the event number and contents for each of the events in the history list except the current event. If *count* is specified, then only the most recent *count* events are returned.

history keep *[count]*

Changes the size of the history list to *count* events. Initially, 20 events are retained in the history list. If *count* is not specified, the current keep limit is returned.

history nextid

Returns the number of the next event to be recorded in the history list. Use this for printing the event number in command-line prompts.

history redo *[event_number]*

Reruns the command indicated by *event* and returns its result. The default value of *event_number* is **-1**. This command results in history revision. See the following section for details.

History Revision

Pre-8.0 Tcl had a complex history revision mechanism. The current mechanism is more limited, and the **substitute** and **words** history operations have been removed. The **clear** operation was added.

The **redo** history option results in much simpler "history revision." When this option is invoked, the most recent event is modified to eliminate the history command and replace it with the result of the history command. If you want to redo an event without modifying the history, use the **event** operation to retrieve an event, and use the **add** operation to add it to history and execute it.

index_collection

Creates a single element collection. I.e. Given a collection and an index into it, if the index is in range, 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
```

Data Types

```
   collection collection1      collection
   int index                   int
```

ARGUMENTS

collection1

Specifies the collection to be searched.

index

Specifies the index into the collection. Allowed values are integers from 0 to **sizeof_collection** - 1.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 shows the first object in a collection being extracted.

```
ptc_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
ptc_shell> query_objects [index_collection $c1 0]
{"u1"}
```

SEE ALSO

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

is_false

Tests the value of a specified variable, and returns 1 if the value is 0 or the case-insensitive string **false**; returns 0 if the value is 1 or the case-insensitive string **true**.

SYNTAX

```
status is_false
   value
```

Data Types

```
   value      string
```

ARGUMENTS

value

Specifies the name of the variable whose value is to be tested.

DESCRIPTION

This command tests the value of a specified variable, and returns 1 if the value is either 0 or the case-insensitive string **false**. The command returns 0 if the value is either 1 or the case-insensitive string **true**. Any value other than 1, 0, **true**, or **false** generates a Tcl error message.

The **is_false** command is used in writing scripts that test Boolean variables that can be set to 1, 0, or the case-insensitive strings **true** or **false**. When such variables are set to **true** or **false**, they cannot be tested in the negative in an **if** statement by simple variable substitution, because they do not constitute a true or false condition. The following example is not legal Tcl:

```
set x FALSE
if { !$x } {
  set y TRUE
}
```

This results in a Tcl error message, indicating that you cannot use a non-numeric string as the operand of "!". So, although you can test the positive condition, **is_false** allows you to test both conditions safely.

EXAMPLES

The following example shows the use of the **is_false** command:

```
prompt> set x TRUE
TRUE

prompt> if { ![is_false $x] } {
?         set y TRUE
?       }
TRUE

prompt>
```

SEE ALSO

expr(2)
if(2)
is_true(2)

is_true

Tests the value of a specified variable, and returns 1 if the value is 1 or the case-insensitive string **true**; returns 0 if the value is 0 or the case-insensitive string **false**.

SYNTAX

```
status is_true
   value
```

Data Types

```
   value      string
```

ARGUMENTS

value

Specifies the name of the variable whose value is to be tested.

DESCRIPTION

This command tests the value of a specified variable, and returns 1 if the value is either 1 or the case-insensitive string **true**. The command returns 0 if the value is either 0 or the case-insensitive string **false**. Any value other than 1, 0, **true**, or **false** generates a Tcl error message.

The **is_true** command is used in writing scripts that test Boolean variables that can be set to 1, 0, or the case-insensitive strings **true** or **false**. When such variables are set to **true** or **false**, they cannot be tested in the negative in an **if** statement by simple variable substitution, because they do not constitute a true or false condition. The following example is not legal Tcl:

```
set x TRUE
if { !$x } {
  set y FALSE
}
```

This results in a Tcl error message, indicating that you cannot use a non-numeric string as the operand of "!". So, although you can test the positive condition, **is_true** allows you to test both conditions safely.

EXAMPLES

The following example shows the use of the **is_true** command:

```
prompt> set x FALSE
FALSE

prompt> if { ![is_true $x] } {
?         set y FALSE
?       }

FALSE

prompt>
```

SEE ALSO

expr(2)
if(2)
is_false(2)

link_design

Resolves references in a design.

SYNTAX

```
string link_design
   [-verbose]
   [-remove_sub_designs]
   [-keep_sub_designs]
   [-add]
   [design_name]
```

Data Types

```
   design_name   string
```

ARGUMENTS

-verbose

Indicates that the linker is to display verbose messages.

-remove_sub_designs

Indicates that subdesigns are to be removed after linking. Use this option to free up memory and improve performance.

-keep_sub_designs

Indicates that subdesigns are to be kept after linking. Use this option to keep the subdesigns around so that current_design can be changed to other designs later.

-add

Indicates that previously linked designs should not be removed. By default, the *link_design* command removes all previously linked designs but keeps resolved references. Use this option to keep previously linked designs around so that current_design can be changed to other designs later.

design_name

Specifies the name of the design to be linked; the default is the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Performs a name-based resolution of design references for the specified *design_name* or the current design. In addition to resolving references in the design, **link_design** builds the internal representation of the design for analysis.

A complete, fully functional design must be connected to all referenced library components and designs; the references must be located and linked to the current design. Thus, the purpose of this command is to locate all designs and library components referenced by the current design and link them to the current design. During the link, all files specified in the **link_path** are loaded if they are not already in memory. The goal is a fully instantiated design on which analysis can be performed.

Automatic Loading of Designs and Libraries

You can set the **link_path** and **search_path** variables so that you need to read only your top design, then link. Constraint consistency automatically finds and loads other designs and libraries that are needed.

In the following example, assume that your main design is in newcpu.db and uses the cmos.db library. The **link_path** variable specifies cmos.db, so the linker loads cmos.db when it starts. As the link proceeds, if a design is required and is not in memory, the linker searches the **search_path** for a file named *reference_name*.db. For example, the referenced BOX1 design is not in memory, so the linker searches for BOX1.db in the **search_path** and loads it.

EXAMPLES

```
ptc_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmos"
/designs/newcpu/v1.6/dbs /libs/cmos
ptc_shell> set link_path "* cmos.db"
* cmos.db
ptc_shell> read_db newcpu.db
Loading db file '/designs/newcpu/v1.6/dbs/newcpu.db'
1
ptc_shell> link_design newcpu
Loading db file '/libs/cmos/cmos.db'
Linking design newcpu...
Loading db file '/designs/newcpu/v1.6/dbs/BOX1.db'
Loading db file '/designs/newcpu/v1.6/dbs/BOX2.db'
Loading db file '/designs/newcpu/v1.6/dbs/padring.db'
Design 'newcpu' was successfully linked.
1
```

Unresolved References

If the linker fails to resolve one or more references, it creates an empty hierarchical cell for each unresolved reference. To fix the problem, first determine and correct the source of the problem, then relink the design using the **link_design** command. Failures are typically caused by missing libraries or designs; incorrectly specified **link_path** or **search_path** variables; or a file that is in the path but is not accessible by you.

If you can resolve the references by changing the **link_path** or **search_path** variables, you must relink the design.

No checking for conflicting references is performed if multiple empty hierarchical cells are created. For example, if SELECT_OP has two references, one with five pins and the other with 20 pins, each of the references leads to the creation of an empty hierarchical cell with the requested number of pins.

EXAMPLES

The following examples show how a design links with many of the different linker options. First, the link fails when a library cannot be found in the link path because of a typo in the search_path.

```
ptc_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmoss"
/designs/newcpu/v1.6/dbs /libs/cmoss
ptc_shell> set link_path "* cmos.db"
* cmos.db
ptc_shell> read_db newcpu.db
Loading db file '/design/newcpu/v1.6/dbs/newcpu.db'
1
ptc_shell> link_design newcpu
Error: Can't read link_path file 'cmos.db' (LNK-801)
Linking design newcpu...
Loading db file '/designs/newcpu/v1.6/dbs/BOX1.db'
Loading db file '/designs/newcpu/v1.6/dbs/BOX2.db'
Loading db file '/designs/newcpu/v1.6/dbs/padring.db'
Warning: Unable to resolve reference to 'NR4' in 'newcpu'. (LNK-805)
Warning: Unable to resolve reference to 'ND2' in 'newcpu'. (LNK-805)
Warning: Unable to resolve reference to 'FD2' in 'newcpu'. (LNK-805)
Information: Design 'newcpu' was not successfully linked:
             16 unresolved references. (LNK-803)
```

In the following example, correcting the value of the **search_path** variable and doing an initial link completely links the design without black boxes.

```
ptc_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmos"
/designs/newcpu/v1.6/dbs /libs/cmos
ptc_shell> link_design newcpu
Loading db file '/libs/cmos/cmos.db'
Linking design newcpu...
Loading db file '/designs/newcpu/v1.6/dbs/BOX1.db'
Loading db file '/designs/newcpu/v1.6/dbs/BOX2.db'
Loading db file '/designs/newcpu/v1.6/dbs/padring.db'
Design 'newcpu' was successfully linked.
1
```

SEE ALSO

```
current_design(2)
read_db(2)
link_path(3)
search_path(3)
```

list_attributes

Lists currently defined attributes.

SYNTAX

```
string list_attributes
   [-application]
   [-class class_name]
   [-nosplit]
```

Data Types

```
   class_name     string
```

ARGUMENTS

-application

Lists application attributes.

-class *class_name*

Limits the listing to attributes of a single class. Valid classes are design, cell, pin, port, net, lib, lib_cell, lib_pin, timing_arc, lib_timing_arc, clock, scenario, input_delay, output_delay, exception, exception_group, clock_group, clock_group_group, rule, ruleset, and rule_violation.

-nosplit

Prevents line splitting and facilitates the writing a tool to extract information from the report output. Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **list_attributes** command displays an alphabetically sorted list of attributes.

The **-application** option should be used to report application attributes. Note that there are many application attributes. It is often useful to limit the listing to a specific object class using the *class_name* option.

Currently, constraint consistency does not support user-defined attributes as constraint consistency does. The **-application** option is used to ensure constraint consistency follows the same use model as the PrimeTime tool.

EXAMPLES

This example limits the listing to net attributes only.

```
ptc_shell> 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
-------------------------------------------------------------------
base_name              net     string     A
full_name              net     string     A
is_global              net     Boolean    A
object_class           net     string     A
```

SEE ALSO

get_attribute(2)
report_attribute(2)

list_libs

Lists libraries that are read into constraint consistency.

SYNTAX

```
string list_libs
   [lib_list]
```

Data Types

```
   lib_list     list
```

ARGUMENTS

lib_list

If specified, the library registry report is restricted to the list of libraries provided in this option.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **list_libs** command lists the libraries that are read into constraint consistency.

An asterisk (*) to the left of a library indicates that this is the main library for the current design. The main library is the first library in the link path which has a time unit.

EXAMPLES

The following example lists all libraries.

```
ptc_shell> list_libs

Library Registry:
    bus1_lib      /u/lib/bus1_lib.db:bus1_lib
    tech1         /u/lib/tech1.db:tech1
```

SEE ALSO

current_design(2)
link_design(2)

list_licenses

Shows the licenses which are currently checked out.

SYNTAX

```
string list_licenses
```

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **list_licenses** command lists the licenses which you currently have checked out. The **list_licenses** command always returns the empty string.

EXAMPLES

This example shows the output from the **list_licenses** command.

```
ptc_shell> list_licenses

Licenses in use:
        GalaxyConstraint (1)
```

SEE ALSO

lminus

Removes one or more named elements from a list and returns a new list.

SYNTAX

```
list lminus
   [-exact]
   original_list
   elements
```

Data Types

```
   original_list     list
   elements          list
```

ARGUMENTS

-exact

Specifies that the exact pattern is to be matched. By default, **lminus** uses the default match mode of **lsearch**, the **-glob** mode.

original_list

Specifies the list to copy and modify.

elements

Specifies a list of elements to remove from *original_list*.

DESCRIPTION

The **lminus** command removes elements from a list by using the element itself, rather than the index of the element in the list (as in **lreplace**). The **lminus** command uses the **lsearch** and **lreplace** commands to find the elements and replace them with nothing.

If none of the elements are found, a copy of *original_list* is returned.

The **lminus** command is often used in the translation of Design Compiler scripts that use the subtraction operator (-) to remove elements from a list.

EXAMPLES

The following example shows the use of the **lminus** command. Notice that no error message is issued if a specified element is not in the list.

```
prompt> set l1 {a b c}
a b c

prompt> set l2 [lminus $l1 {a b d}]
c

prompt> set l3 [lminus $l1 d]
a b c
```

The following example illustrates the use of **lminus** with the **-exact** option:

```
prompt> set l1 {a a[1] a* b[1] b c}
a a[1] a* b[1] b c

prompt> set l2 [lminus $l1 a*]
{b[1]} b c

prompt> set l3 [lminus -exact $l1 a*]
a {a[1]} {b[1]} b c

prompt> set l4 [lminus -exact $l1 {a[1] b[1]} ]
a a* b c
```

SEE ALSO

```
lreplace(2)
lsearch(2)
```

load_of

Gets the capacitance of a library cell pin. It is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
float load_of
   lib_pin
```

Data Types

```
   lib_pin       string
```

ARGUMENTS

lib_pin

Specifies the name of the library cell pin, or a collection that contains the library cell pin, for which to get the capacitance.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **load_of** command returns the capacitance of the given library cell pin. The command exists in constraint consistency for compatibility with Design Compiler. Complete information about the **load_of** command can be found in the Design Compiler documentation. The supported method for getting the capacitance of a library cell pin is by using the **get_attribute** command with the **pin_capacitance** attribute.

SEE ALSO

get_attribute(2)

ls

Lists the contents of a directory.

SYNTAX

```
string ls [filename ...]

string filename
```

ARGUMENTS

filename

Provides the name of a directory or filename, or a pattern which matches files or directories.

DESCRIPTION

If no argument is specified, the contents of the current directory are listed. For each *filename* matching a directory, **ls** lists the contents of that directory. If *filename* matches a file name, the file name is listed.

EXAMPLES

```
shell> ls *.db *.pt

test1.pt         c1.db          c3.db          c5.db

test2.pt         c2.db          c4.db          c6.db
```

SEE ALSO

cd(2)
pwd(2)

man

Displays reference manual pages.

SYNTAX

```
string man [-text_only] topic
   [-text_only]
   topic
```

Data Types

```
   topic     string
```

ARGUMENTS

-text_only

Shows only text version of the man page for the specified command. Does not launch the online help browser.

topic

Specifies the subject to display. Available topics include commands, variables, and error messages,

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **man** command displays the online man page for a command, variable, or error message. You can write man pages for your own Tcl procedures and access them with the **man** command by setting the **sh_user_man_path** variable to an appropriate value. See the man page for **sh_user_man_path** for more information.

EXAMPLES

The following command displays the man page for the **echo** command.

```
shell> man echo
```

The following command displays the man page for the error message CMD-025.

```
shell> man CMD-025
```

SEE ALSO

help(2)
sh_user_man_path(3)

mem

Retrieves the total memory allocated by the current ptc_shell process.

SYNTAX

```
int mem
```

ARGUMENTS

The **mem** command has no arguments.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The *mem* command returns the size of memory currently allocated by the current ptc_shell process for the purposes of storing design netlist, design annotations, and constraints information, as well as computed timing information. The value printed represents the amount in kilobytes (KB).

Note that the value reported would not match values obtained by the user through UNIX process tracking commands, such as *top*. This is the case because the *mem* command does not track memory used to load the executable and maintain process stack space. Also, it does not differentiate between resident and swapped memory.

EXAMPLES

```
ptc_shell> mem
8092
```

SEE ALSO

cputime(2)

parse_proc_arguments

Parses the arguments passed into a Tcl procedure.

SYNTAX

```
string parse_proc_arguments -args arg_list
result_array

list arg_list
string result_array
```

ARGUMENTS

-args *arg_list*

Specified the list of arguments passed in to the Tcl procedure.

result_array

Specifies the name of the array into which the parsed arguments should be stored.

DESCRIPTION

The **parse_proc_arguments** command is used within a Tcl procedure to enable use of the -help option, and to support argument validation. It should typically be the first command called within a procedure. Procedures that use **parse_proc_arguments** will validate the semantics of the procedure arguments and generate the same syntax and semantic error messages as any application command (see the examples that follow).

When a procedure that uses **parse_proc_arguments** is invoked with the -help option, **parse_proc_arguments** will print help information (in the same style as using **help -verbose**) and will then cause the calling procedure to return. Similarly, if there was any type of error with the arguments (missing required arguments, invalid value, and so on), **parse_proc_arguments** will return a Tcl error and the calling procedure will terminate and return.

If you didn't specify -help, and the specified arguments were valid, the array variable *result_array* will contain each of the argument values, subscripted with the argument name. Note that the argument name here is NOT the names of the arguments in the procedure definition, but rather the names of the arguments as defined using the **define_proc_attributes** command.

The **parse_proc_arguments** command cannot be used outside of a procedure.

EXAMPLES

The following procedure shows how **parse_proc_arguments** is typically used. The argHandler procedure parses the arguments it receives. If the parse is successful, argHandler prints the options or values actually received.

```
proc argHandler args {
  parse_proc_arguments -args $args results
  foreach argname [array names results] {
    echo "  $argname = $results($argname)"
  }
}
define_proc_attributes argHandler -info "argument processor" \
  -define_args \
  {{-Oos "oos help" AnOos one_of_string {required value_help {values {a b}}}}
   {-Int "int help" AnInt int optional}
   {-Float "float help" AFloat float optional}
   {-Bool "bool help" "" boolean optional}
   {-String "string help" AString string optional}
   {-List "list help" AList list optional}}
  {-IDup int dup AIDup int {optional merge_duplicates}}}
```

Invoking argHandler with the -help option generates the following:

```
prompt> argHandler -help
Usage: argHandler    # argument processor
        -Oos AnOos              (oos help:
                                 Values: a, b)
        [-Int AnInt]            (int help)
        [-Float AFloat]         (float help)
        [-Bool]                 (bool help)
        [-String AString]       (string help)
        [-List AList]           (list help)
```

Invoking argHandler with an invalid option causes the following output (and a Tcl error):

```
prompt> argHandler -Int z
Error: value 'z' for option '-Int' not of type 'integer' (CMD-009)
Error: Required argument '-Oos' was not found (CMD-007)
```

Invoking argHandler with valid arguments generates the following:

```
prompt> argHandler -Int 6 -Oos a
  -Oos = a
  -Int = 6
```

SEE ALSO

define_proc_attributes(2)
help(2)
proc(2)

print_message_info

Prints information about diagnostic messages that have occurred or have been limited.

SYNTAX

```
string print_message_info
   [-ids id_list]
   [-summary]
```

Data Types

```
   id_list              list
```

ARGUMENTS

-ids *id_list*

List of message identifiers to report. Each entry can be a specific message or a glob-style pattern that matches one or more messages. If this option is omitted and no other options are given, all of the messages that have occurred or have been limited are reported.

-summary

Generate a summary of error, warning, and informational messages that have occurred so far.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **print_message_info** command enables you to print summary information about error, warning, and informational messages that have occurred or have been limited with the **set_message_info** command. For example, if the following message is generated, information about it is recorded:

Error: unknown command 'wrong_command' (CMD-005)

It is useful to be able to summarize all recorded information about generated diagnostic messages. Much of this can be done using the **get_message_info** command, but you need to know the specific message ID. By default, the **print_message_info** command summarizes all of the information. It provides a single line for each message that has occurred or has been limited, and one summary line that shows the total number of errors, warnings, and informational messages that have occurred so far. If an *id_list* is given, only messages matching those pat- terns are displayed. If the *-summary* option is given, a summary is displayed.

Using a pattern in the *id_list* is intended to show a specific message prefix, for example, "CMD*". Note that this does not show all messages with that prefix. Only those that have occurred or have been limited are displayed. By default, the limit column shows the limit set by the **set_message_info** command. However, if the limit is not set and this type of message is included in the **sh_limited_messages** variable, this column shows the default limit of the **sh_message_limit** variable, and a symbol '*' also appears to show that this limit is from the **sh_message_limit** variable.

EXAMPLES

The following example uses the **print_message_info** command to display a few specific messages.

```
shell> print_message_info -ids [list "CMD*" APP-99]

  Id             Severity         Limit    Occurrences   Suppressed
  ------------------------------------------------------------------
  CMD-005        Error            0        7             2
  APP-99         Information      1        0             0
```

At the end of the session, you might want to generate some information about a set of interesting messages, such as how many times each occurred (which includes suppressions), how many times each was suppressed, and whether a limit was set for any of them. The following example shows how to use the **print_message_info** command in this way. The symbol '*' in the limit column of PARA-004 means that this message is in the **sh_limited_messages** variable and this limit is the **sh_message_limit** variable.

```
shell> print_message_info

 Id             Severity         Limit      Occurrences   Suppressed
 ------------------------------------------------------------------
 CMD-005        Error              0          12            0
 APP-027        Information      100         150           50
 APP-99         Information        0           1            0

 Diagnostics summary: 12 errors, 151 warnings, 1 informational
```

Note that the suppressed count is not necessarily the difference between the limit and the occurrences, since the limit can be dynamically changed with the **set_message_info** command, or the limit is from the **sh_message_limit** variable.

The sorting is by Severity (Error, Warning, Information), then by Occurrences, in descending order, then by Id.

SEE ALSO

```
get_message_info(2)
set_message_info(2)
suppress_message(2)
sh_message_limit(3)
sh_limited_messages(3)
```

print_suppressed_messages

Displays an alphabetical list of message IDs that are currently suppressed.

SYNTAX

```
string print_suppressed_messages
```

ARGUMENTS

The **print_suppressed_messages** command has no arguments.

DESCRIPTION

The **print_suppressed_messages** command displays all messages that you suppressed using the **suppress_message** command. The messages are listed in alphabetical order. You only can suppress informational and warning messages. The result of **print_suppressed_messages** is always the empty string.

EXAMPLES

The following example shows the output from the **print_suppressed_messages** command:

```
prompt> print_suppressed_messages
No messages are suppressed

prompt> suppress_message {XYZ-001 CMD-029 UI-1}

prompt> print_suppressed_messages
The following 3 messages are suppressed:
```

```
CMD-029, UI-1, XYZ-001
```

SEE ALSO

```
suppress_message(2)
unsuppress_message(2)
```

printenv

Prints the value of environment variables.

SYNTAX

```
string printenv
   [variable_name]
```

Data Types

```
   variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of a single environment variable to print.

DESCRIPTION

The **printenv** command prints the values of the environment variables inherited from the parent process or set from within the application with the **setenv** command. When no arguments are specified, all environment variables are listed. When a single *variable_name* is specified, if that variable exists, its value is printed. There is no useful return value from **printenv**.

To retrieve the value of a single environment variable, use the **getenv** command.

EXAMPLES

The following examples show the output from the **printenv** command:

```
prompt> printenv SHELL
/bin/csh

prompt> printenv EDITOR
emacs
```

SEE ALSO

getenv(2)
setenv(2)

printvar

Prints the values of one or more variables.

SYNTAX

```
string printvar
   [pattern]
   [-user_defined | -application]
```

Data Types

```
   pattern      string
```

ARGUMENTS

pattern

Prints variable names that match *pattern*. The optional *pattern* argument can include the wildcard characters * (asterisk) and ? (question mark). If not specified, all variables are printed.

-user_defined

Shows only user-defined variables. This option is mutually exclusive with the **-application** option.

-application

Shows only application variables. This option is mutually exclusive with the **-user_defined** option.

DESCRIPTION

The **printvar** command prints the values of one or more variables. If the *pattern* argument is not specified, the command prints out the values of application and user-defined variables. The **-user_defined** option limits the variables to those that you have defined. The **-application** option limits the variables to those created by the application. The two options are mutually exclusive and cannot be used together.

Some variables are suppressed from the output of **printvar**. For example, the Tcl environment variable array *env* is not shown. To see the environment variables, use the **printenv** command. Also, the Tcl variable **errorInfo** is not shown. To see the error stack, use the **error_info** command.

EXAMPLES

The following command prints the values of all of the variables:

```
prompt> printvar
```

The following command prints the values of all application variables that match the pattern *sh*a**:

```
prompt> printvar -application sh*a*
sh_arch                 = "sparcOS5"
sh_command_abbrev_mode  = "Anywhere"
sh_enable_page_mode     = "false"
sh_new_variable_message = "false"
sh_new_variable_message_in_proc = "false"
sh_source_uses_search_path = "false"
```

The following command prints the value of the **search_path** variable:

```
prompt> printvar search_path
search_path             = ".  /designs/newcpu/v1.6  /lib/cmos"
```

The following command prints the values of the user-defined variables:

```
prompt> printvar -user_defined
a                   = "6"
designDir           = "/u/designs"
```

SEE ALSO

```
error_info(2)
printenv(2)
setenv(2)
```

proc_args

Displays the formal parameters of a procedure.

SYNTAX

```
string proc_args
   proc_name
```

Data Types

```
   proc_name      string
```

ARGUMENTS

proc_name

Specifies the name of the procedure.

DESCRIPTION

The **proc_args** command is used to display the names of the formal parameters of a user-defined procedure. This command is essentially a synonym for the Tcl builtin command **info** with the *args* argument.

EXAMPLES

This example shows the output of **proc_args** for a simple procedure:

```
prompt> proc plus {a b} { return [expr $a + $b] }
```

```
prompt> proc_args plus
a b

prompt> info args plus
a b

prompt>
```

SEE ALSO

info(2)
proc(2)
proc_body(2)

proc_body

Displays the body of a procedure.

SYNTAX

```
string proc_body
   proc_name
```

Data Types

```
   proc_name      string
```

ARGUMENTS

proc_name

Specifies the name of the procedure.

DESCRIPTION

The **proc_body** command is used to display the body (contents) of a user-defined procedure. This command is essentially a synonym for the Tcl builtin command **info** with the *body* argument.

EXAMPLES

This example shows the output of **proc_body** for a simple procedure:

```
prompt> proc plus {a b} { return [expr $a + $b] }

prompt> proc_body plus
```

```
 return [expr $a  + $ b]

prompt> info body plus
 return [expr $a  + $ b]

prompt>
```

SEE ALSO

info(2)
proc(2)
proc_args(2)

query_objects

Searches for and displays objects in the database.

SYNTAX

```
string query_objects [-verbose]
[-class class_name]


   [-truncate elem_count]
   object_spec

string  class_name
int     elem_count
list    object_spec
```

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 preceeded 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, and so on.

-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

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **query_objects** command (a simple interface) finds and displays objects in the constraint consistency 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** simply displays the contents of the collection.

For elements of the *object_spec* that are names (or wildcarded patterns), **query_objects** searches for the objects in the class specified by *class_name*. Note that **query_objects** 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 (see EXAMPLES).

To control the number of elements displayed, use the *-truncate* option. If the display is truncated, you see the elipsis (...) as the last element. Note that if the default truncation occurs, a message displays showing the total number of elements that would have displayed (see EXAMPLES).

NOTE: The output from **query_objects** looks similar to the output from any command which creates a collection, but remember that the result of **query_objects** is always the empty string.

EXAMPLES

These following examples show the basic usage of **query_objects**.

```
ptc_shell> query_objects [get_cells o*]
{"or1", "or2", "or3"}
ptc_shell> query_objects -class cell U*
{"U1", "U2"}
ptc_shell> 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.

```
ptc_shell> 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 elipsis at the end of the display. For the following example, assume the default truncation is 5 (it is actually 100).

```
ptc_shell> query_objects [get_cells o*] -truncate 2
{"or1", "or2", ...}
ptc_shell> 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)

quit

Exits the shell.

SYNTAX

```
string quit
```

ARGUMENTS

The **quit** command has no arguments.

DESCRIPTION

The **quit** command exits from the application. It is basically a synonym for the **exit** command with no arguments.

EXAMPLES

The following example exits the current session:

```
prompt> quit
```

SEE ALSO

exit(2)

read_db

Reads in one or more library files in the Synopsys database (db) format.

SYNTAX

```
string read_db
   file_names
```

Data Types

```
   file_names    list
```

ARGUMENTS

file_names

Specifies names of one or more files to be read. Typically, only one file is read.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reads library information from the Synopsys database (db) files into constraint consistency.

To locate files that have relative pathnames, the **read_db** command uses the **search_path** variable to search for each file in each directory listed in the **search_path**. Files that have absolute path names are loaded as though there were no **search_path**. To determine the file that **read_db** loads, use the **which** command.

Each file can contain one cell library. The db file itself is used only during the read process and is transformed into the internal format of constraint consistency during the **read_db** command. To remove libraries, use the **remove_lib** command.

Note that libraries are read automatically during the **link_design** command, or implicit linking based on the **link_path** and **search_path** variables, so it is usually not necessary to use the **read_db** command.

EXAMPLES

In the following example, the file mylib.db is read based on the *search_path*.

```
ptc_shell> set search_path "/libs/mylib/v1.6/dbs /libs/cmos"

ptc_shell> read_db mylib.db
Loading db file '/libs/mylib/v1.6/dbs/mylib.db'
1
```

SEE ALSO

link_design(2)
remove_lib(2)
which(2)
link_path(3)
search_path(3)

read_file

Reads a netlist or library file. This is a DC Emulation command provided for compatibility with the Design Compiler tool.

SYNTAX

```
string read_file
   [-format type]
   [-single_file ns1]
   [-names_file ns2]
   [-define ns3]
   file_list
```

Data Types

```
   type           string
   ns1            string
   ns2            string
   ns3            string
   file_list      list
```

ARGUMENTS

-format *type*

Specifies the file format. Allowed values in constraint consistency are *db* and *verilog*.

-single_file *ns1*

Not supported by constraint consistency.

-names_file *ns2*

Not supported by constraint consistency.

-define *ns3*

Not supported by constraint consistency.

file_list

A list of files that are to be read.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **read_file** command reads in one or more netlist or library files. The command exists in constraint consistency for compatibility with the PrimeTime and Design Compiler tools. Complete information about the **read_file** command can be found in the Design Compiler documentation. The supported method for reading netlist or library files is to use the following commands: **read_db**, **read_verilog**.

SEE ALSO

```
read_db(2)
read_verilog(2)
```

read_sdc

Reads in a script in the Synopsys Design Constraints (SDC) format.

SYNTAX

```
int read_sdc file_name [-echo]
   [-echo]
   [-syntax_only]
   [-version sdc_version]
```

Data Types

```
   file_name     string
   sdc_version   string
```

ARGUMENTS

-echo

Indicates that each constraint is to be echoed as it is executed.

-syntax_only

Indicates that the SDC script is processed only to check the syntax and semantics of the script.

-version *sdc_version*

Specifies the version of the SDC script to which the file conforms. Allowed values are *1.0*, *1.1*, *1.2*, *1.3*, *1.4*, *1.5*, *1.6*, and *latest* (the default).

file_name

Specifies the name of the file that contains the SDC script to be read.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **read_sdc** command reads in a script file in Synopsys Design Constraints (SDC) format, which contains commands for use by constraint consistency or by Design Compiler in its Tcl mode. The SDC format is also licensed by external vendors through the Tap-in program. The SDC-formatted script files are Tcl scripts that use a subset of the commands supported by constraint consistency and Design Compiler.

By default, the **read_sdc** command executes the constraints as they are read. If there are errors halfway through the script, it is possible that some constraints are applied, and some are not. Use the **-syntax_only** option to check the script without applying any constraints. This also verifies conformance to the specified SDC version. If there are any errors during the checking phase, the result of **read_sdc** is 0.

Like the **source** command, the **read_sdc** command is sensitive to variables that control script execution (for example, **sh_continue_on_error** and **sh_script_stop_severity**). The **read_sdc** command uses the **sh_source_uses_search_path** variable to determine if the **search_path** variable is used to find the script.

read_sdc command supports several file formats. The file can be a simple ASCII script file, an ASCII script file compressed with gzip, or a Tcl bytecode script, created by TclPro Compiler.

Note that the SDC format does not allow command abbreviation. Also note that the **exit** and **quit** command do not cause the application to exit, but they do cause the reading of the SDC file to stop.

The latest SDC version supports the following commands. Those added for versions 1.3 and later are noted. Some constraints which constraint consistency ignores are not listed.

```
General Purpose Commands:
list
expr
set

Object Access Functions:
all_clocks
all_inputs
all_outputs
current_design
current_instance
get_cells
get_clocks
get_libs
get_lib_cells
get_lib_pins
get_nets
get_pins
get_ports
set_hierarchy_separator

Basic Timing Assertions:
create_clock
create_generated_clock          (1.3)
set_clock_gating_check
set_clock_latency
set_clock_transition
set_clock_uncertainty
set_data_check                  (1.4)
set_false_path
set_input_delay
set_max_delay
set_min_delay
set_multicycle_path
set_output_delay
```

```
set_propagated_clock

Secondary Assertions:
set_disable_timing
set_max_time_borrow
set_timing_derate                (1.5)

Environment Assertions:
set_case_analysis
set_drive
set_driving_cell
set_fanout_load
set_input_transition
set_load
set_logic_zero
set_logic_one
set_logic_dc
set_max_area
set_max_capacitance
set_max_fanout
set_max_transition
set_min_capacitance
set_operating_conditions
set_port_fanout_number
set_resistance
set_wire_load_min_block_size
set_wire_load_mode
set_wire_load_model
set_wire_load_selection_group
```

For a complete guide to using the SDC format with Synopsys applications, see the *Using the Synopsys Design Constraints Format Application Note* in Documentation on the Web, which is available through SolvNet at http://solvnet.synopsys.com.

The usage of some of these commands is restricted when reading the SDC format. In certain cases, some command options are not allowed. In certain applications, some of the commands are not supported. For example, constraint consistency does not support the **set_logic*** commands. These commands are ignored when loaded with the **read_sdc** command, with a message (for an example, see the EXAMPLES section). If a command that is not supported by the SDC format is found by the **read_sdc** command, it appears as an unknown command, with a message. The same condition exists for command options. If an option is not supported by the SDC format, a message is issued indicating that condition. However, if the option is unknown, a different message is issued.

Note that although the **create_generated_clock** command is supported (beginning with version 1.3), there is no provision for the **get_generated_clocks** command shortcut, which is available in constraint consistency and Design Compiler. Generated clocks are simply clocks with some additional attributes, and in the SDC format, they are retrieved through the **get_clocks** command.

The following features are added for SDC version 1.5 and later: **set_clock_latency -clock** option, **set_timing_derate** command with all options, **set_max_transition -clock_path -data_path** -rise -fall options, **set_clock_uncertainty -rise/fall_from/to** options, **set_operating_conditions** -object_list option, synonyms for all **get_object** commands, **get_objects -nocase** support independently with -regexp, **get_objects -regexp** support, **get_objects -of_objects** support for **get_cells/get_nets/get_pins**.

EXAMPLES

The following example reads the SDC script, top.sdc:

```
ptc_shell> read_sdc top.sdc -version 1.2
Reading SDC version 1.2...
1
ptc_shell>
```

The following example reads the SDC script, top2.sdc. The script contains commands that are not supported by the SDC format, and CMD-005 messages are generated for those commands.

```
ptc_shell> read_sdc -syntax_only top2.pt -version 1.2

Checking syntax/semantics using SDC version 1.2...

Error: Unknown command 'link_design' (CMD-005)

** Completed syntax/semantic check with 1 error(s) **
0
ptc_shell>
```

The following example shows the result when an unsupported command is in an SDC file. One SDC message is generated per instance of an unsupported command. A summary of all unsupported commands in the file is generated. This SDC file was read into constraint consistency.

```
ptc_shell> read_sdc t2.sdc -version 1.2
Reading SDC version 1.2...
Warning: Constraint 'set_logic_zero' is not supported by ptc_shell. (SDC-3)
Warning: Constraint 'set_logic_zero' is not supported by ptc_shell. (SDC-3)
Warning: Constraint 'set_logic_one' is not supported by ptc_shell. (SDC-3)

Summary of unsupported constraints:
Information: Ignored 1 unsupported 'set_logic_one' constraint. (SDC-4)
Information: Ignored 2 unsupported 'set_logic_zero' constraints. (SDC-4)

1
ptc_shell>
```

The following example is a script that is not SDC-compliant, because it contains unsupported commands or command options.

```
current_design TOP
set_resistance -value 12.2 [get_nets i1t2]
```

The following example shows the output generated by the **read_sdc** command when reading the previous script. In the SDC format, the **current_design** command is used only to obtain the current design; no arguments are allowed. Also, there is no **-value** option for the **set_resistance** command, so an appropriate message is generated.

```
ptc_shell> read_sdc t3.tcl -echo -version 1.2
Reading SDC version 1.2...
current_design TOP
Error: extra positional option 'TOP' (CMD-012)
set_resistance -value 12.2 [get_nets i1t2]
Error: unknown option '-value' (CMD-010)
```

```
0
ptc_shell>
```

The following example shows the message generated when an option is supported by the command but not by the SDC format.

```
Information: The -value option for set_resistance is unsupported. (CMD-038)
```

SEE ALSO

```
source(2)
write_sdc(2)
search_path(3)
sh_continue_on_error(3)
sh_script_stop_severity(3)
sh_source_uses_search_path(3)
```

read_verilog

Reads in one or more Verilog files.

SYNTAX

```
string read_verilog file_names

list file_names
```

ARGUMENTS

file_names

Specifies names of one or more files to be read.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reads one or more structural, gate-level Verilog netlists into constraint consistency. To locate files that have relative pathnames, the command uses the **search_path** variable and searches for each file in each directory listed in the **search_path**. Files that have absolute pathnames are loaded as though there were no **search_path**. To determine the file that **read_verilog** loads, use the **which** command. After the Verilog files are loaded, to view the design objects, use the **list_designs** command. To remove designs, use the **remove_design** command.

The Verilog netlist must contain fully-mapped, structural designs. Constraint consistency cannot link or perform timing analysis with netlists that are not fully mapped at the gate level. There must be no Verilog high-level constructs in the netlist.

The Verilog file can be a simple ascii file, or it can be compressed with gzip. Files which are compressed with gzip are first uncompressed to a temporary file in the directory stored in the variable **gca_tmp_dir**.

Generally, the Verilog reader will not create unconnected nets. If you need to have these nets in your design, set the variable **svr_keep_unconnected_nets** to true. Also, if a module has one or more

references to a reference name but no instances have connections, those instances will be omitted if the variable **svr_keep_unconnected_cells** is set to "false" (the default). This saves memory for instances of filler cells and other cells with no pins.

You can use the **bus_naming_style** variable to control bus naming.

Limitations of Constraint Consistency Verilog Reader

The Verilog reader in constraint consistency has the following limitations:

- No non-structural constructs are supported. For details, see the

  section entitled "Supported Language Subset for Constraint Consistency Verilog Reader".

- Parameters are not supported. The parameter and defparam statements are read and ignored.
- Gate instantiations are not supported.

Supported Language Subset for Constraint Consistency Verilog Reader

The Verilog reader supports a small structural subset of the Verilog language. Additional constructs are recognized and ignored. Supported constructs are as follows:

- module/endmodule
- input, inout, and output
- wire
- tri, wor, and wand: These constructs are supported, but are considered

  to be the same as the wire construct. That is, there is no special significance to these constructs.

- supply0 and supply1: These create wires that are logic 0 and 1,

  respectively.

- assign
- tran: An exception from the gate instantiation subset, tran is

  supported to the extent that it relates one wire to another, as with assign. Beyond that, tran has no special significance.

- The preprocessor directives `define, `undef, `include, `ifdef, `else,

  and `endif are not supported by default, as the target for this reader is structural, machine generated Verilog, which rarely contains these constructs. You can enable a preprocessor phase by setting the variable **svr_enable_vpp** to true. This will make an pre-pass over the file looking for and expanding these constructs, outputting a temporary file in the directory stored in the variable **gca_tmp_dir**.

- All simulation directives (for example, `timescale)

  are recognized and ignored.

- The specify/endspecify construct and its contents

  are recognized and ignored.

- Like HDL Compiler, the comment directives *translate_off*

and *translate_on* are used to bypass Verilog features not supported by the reader. The following example shows how to bypass an unsupported feature:

```
// synopsys translate_off
nand (n2, a1, s2);
/* synopsys translate_on */
```

EXAMPLES

In the following example, the file newcpu.v is read based on the **search_path**.

```
ptc_shell> set search_path "/designs/newcpu/v1.6 /libs/cmos"
/designs/newcpu/v1.6 /libs/cmos

ptc_shell> read_verilog newcpu.v
Loading verilog file '/designs/newcpu/v1.6/newcpu.v'
1
```

SEE ALSO

remove_design(2)
which(2)
bus_naming_style(3)
gca_tmp_dir(3)
search_path(3)
svr_enable_vpp(3)
svr_keep_unconnected_cells(3)
svr_keep_unconnected_nets(3)

redirect

Redirects the output of a command to a file.

SYNTAX

```
string redirect
[-append] [-tee] [-file | -variable | -channel] [-compress]
target
{command_string}

string target
string command_string
```

ARGUMENTS

-append

Appends the output to *target*.

-tee

Like the unix command of the same name, sends output to the current output channel as well as to the *target*.

-file

Indicates that *target* is a file name, and redirection is to that file. This is the default. It is exclusive of *-variable* and *-channel*.

-variable

Indicates that *target* is a variable name, and redirection is to that Tcl variable. It is exclusive of *-file* and *-channel*.

-channel

Indicates tha *target* is a Tcl channel, and redirection is to that channel. It is exclusive of *-variable* and *-file*.

-compress

Compress when writing to file. If redirecting to a file, then this option specifies that the output should be

compressed as it is written. The output file is in a format recognizable by "gzip -d". You cannot specify this option with *-append*.

target

Indicates the target of the output redirection. This is either a filename, Tcl variable, or Tcl channel depending on the command arguments.

command_string

The command to execute. Intermediate output from this command, as well as the result of the command, will be redirected to *target*. The *command_string* should be rigidly quoted with curly braces.

DESCRIPTION

The **redirect** command performs the same function as the traditional unix-style redirection operators > and >>. The *command_string* must be rigidly quoted (that is, enclosed in curly braces) in order for the operation to succeed.

Output is redirected to a file by default. Output can be redirected to a Tcl variable by using the *-variable* option, or to a Tcl channel by using the *-channel* option.

Output can be sent to the current output device as well as the redirect target by using the *-tee* option. See the examples section for an example.

The result of a **redirect** command which does not generate a Tcl error is the empty string. Screen output occurs only if errors occurred during execution of the *command_string* (other than opening the redirect file). When errors occur, a summary message is printed. See the examples.

Although the result of a successful **redirect** command is the empty string, it is still possible to get and use the result of the command that you redirected. Construct a **set** command in which you set a variable to the result of your command. Then, redirect the **set** command. The variable holds the result of your command. See the examples.

The **redirect** command is much more flexible that traditional unix redirection operators. With **redirect**, you can redirect multiple commands or an entire script. See the examples for an example of how to construct such a command.

Note that the builtin Tcl command **puts** does not respond to output redirection of any kind. Use the builtin **echo** command instead.

EXAMPLES

In the following example, the output of the plus procedure is redirected. The echoed string and the result of the plus operation is in the output file. Notice that the result was not echoed to the screen.

```
prompt> proc plus {a b} {echo "In plus" ; return [expr $a + $b]}
```

```
prompt> redirect p.out {plus 12 13}
prompt> exec cat p.out
In plus
25
```

In this example, a typo in the command created an error condition. The error message indicates that you can use **error_info** to trace the error, but you should first check the output file.

```
prompt> redirect p.out {plus2 12 13}
Error: Errors detected during redirect
        Use error_info for more info. (CMD-013)
prompt> exec cat p.out
Error: unknown command 'plus2' (CMD-005)
```

In this example, we explore the usage of results from redirected commands. Since the result of **redirect** for a command which does not generate a Tcl error is the empty string, use the **set** command to trap the result of the command. For example, assume that there is a command to read a file which has a result of "1" if it succeeds, and "0" if it fails. If you redirect only the command, there is no way to know if it succeeded.

```
redirect p.out { read_a_file "a.txt" }
# Now what?  How can I redirect and use the result?
```

But if you set a variable to the result, then it is possible to use that result in a conditional expression, etc.

```
redirect p.out { set rres [read_a_file "a.txt"] }
if { $rres == 1 } {
  echo "Read ok!"
}
```

The **redirect** command is not limited to redirection of a single command. You can redirect entire blocks of a script with a single **redirect** command. This simple example with **echo** demonstrates this feature:

```
prompt> redirect p.out {
?         echo -n "Hello "
?         echo "world"
?       }
prompt> exec cat p.out
Hello world
prompt>
```

The **redirect** command allows you to tee output to the previous output device and also to redirect output to a variable. This simple example with **echo** demonstrates these features:

```
prompt> set y "This is "
This is
prompt> redirect -tee x.out {
          echo XXX
          redirect -variable y -append {
            echo YYY
            redirect -tee -variable z {
```

```
                echo ZZZ
              }
            }
          }
XXX
prompt> exec cat x.out
XXX
prompt> echo $y
This is YYY
ZZZ

prompt> echo $z
ZZZ
```

SEE ALSO

echo(2)
error_info(2)
set(2)

remove_annotated_transition

Removes previously-annotated transition times from pins or ports in the current design.

SYNTAX

```
int remove_annotated_transition
   -all | pin_or_port_list
```

Data Types

```
   pin_or_port_list         list
```

ARGUMENTS

-all

Indicates that all annotated transition times in the design are to be removed. The *-all* and *pin_list* options are mutually exclusive. You must set one of these options, but not both.

pin_or_port_list

Specifies a list of pins or ports from which annotated transition times are to be removed. The *-all* and *pin_list* options are mutually exclusive. You must set one of these options, but not both.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_annotated_transition** command removes transition times previously annotated on pins or ports from the current design using the **set_annotated_transition** command.

There are two ways to use the **remove_annotated_transition** command.

- You can remove all annotated transition times from the entire design using the *-all* option.

- You can remove all annotated transition times from a list of pins or ports using the *pin_list* option.

EXAMPLES

The following example removes all annotated pin and port transition times from the current design.

```
ptc_shell> remove_annotated_transition -all
1
```

The following example removes annotated transition time from the input pin **A** of cell "U1/U2/U3".

```
ptc_shell> remove_annotated_transition [get_pins U1/U2/U3/A]
1
```

SEE ALSO

```
reset_design(2)
set_annotated_transition(2)
```

remove_capacitance

Removes capacitance on nets or ports.

SYNTAX

```
string remove_capacitance
   net_or_port_list
```

Data Types

```
   net_or_port_list      list
```

ARGUMENTS

net_or_port_list

Specifies a list of ports and nets in the current design, whose capacitances are removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies that the lumped capacitance annotated on the list of nets or ports must be removed.

Annotated capacitances are also removed on the full design by doing a *reset_design*.

EXAMPLES

The following example removes the annotated capacitance on net w23.

```
ptc_shell> remove_capacitance [get_nets "w23"]
```

The following example removes the annotated capacitance on all ports that name match "P2*".

```
ptc_shell> remove_capacitance [get_ports "P2*"]
```

SEE ALSO

set_load(2)

remove_case_analysis

Removes the case analysis value on input.

SYNTAX

```
string remove_case_analysis

[-all]
[port_or_pin_list]
```

ARGUMENTS

-all

Specifies to remove all user case values in the current design.

port_or_pin_list

Lists ports or pins for which the case analysis entry is to be removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command removes previously specified entries of case analysis on ports or pins. Case analysis is specified using command **set_case_analysis**. The command specifies that a pin or port is at a constant logic value (0 or 1), or that only one of the rising or falling transition is valid. You must specify either the *port_or_pin_list* or **-all** options.

EXAMPLES

The following example specifies that ports in0 , in1 and in2 are set to the constant logic value 0.

ptc_shell> **set_case_analysis 0 {in0 in1 in2}** ptc_shell> **report_case_analysis**

```
*************************************
Report : case_analysis
Design : middle
Version: ...
Date   : Mon Jul 22 08:13:50 1996
*************************************

Pin name                                          Case analysis value
-------------------------------------------------------------------------------
in2                                               0
in1                                               0
in0                                               0
```

The following example removes the case analysis entry on port in2.

ptc_shell> **remove_case_analysis {in2}** ptc_shell> **report_case_analysis**

```
*************************************
Report : case_analysis
Design : middle
Version: ...
Date   : Mon Jul 22 08:14:16 1996
*************************************

Pin name                                          Case analysis value
-------------------------------------------------------------------------------
in1                                               0
in0                                               0
```

The following example removes all user case values from the current design.

ptc_shell> **remove_case_analysis -all** ptc_shell> **report_case_analysis**

```
*************************************
Report : case_analysis
Design : middle
Version: ...
Date   : Mon Jul 22 08:14:16 1996
*************************************

Pin name                                          Case analysis value
-------------------------------------------------------------------------------
```

SEE ALSO

set_case_analysis(2)
report_case_analysis(2)

remove_clock

Removes one or more clocks from the current design.

SYNTAX

```
string remove_clock
   [-all]
   [clock_list]
```

Data Types

```
   clock_list     list
```

ARGUMENTS

-all

Specifies to remove all clocks in the current design.

clock_list

Specifies a list of collections containing clocks or patterns matching the clock names.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes the specified clock objects from the current design. You must specify either the *clock_list* or **-all** options.

If a removed clock is the only member of a path group, the path group is also removed. For information about path groups, refer to the **group_path** command man page. To list all clock sources in the design, use the **report_clock** command.

All arrival and required times specified by the **set_input_delay** and **set_output_delay** commands

relative to the clock that is being removed are also removed.

EXAMPLES

The following example removes clock CLK1.

```
ptc_shell> remove_clock CLK1
```

The following example removes all clocks from the current design.

```
ptc_shell> remove_clock -all
```

SEE ALSO

create_clock(2)
current_design(2)
get_clocks(2)
group_path(2)
report_clock(2)
set_input_delay(2)
set_output_delay(2)
reset_design(2)

remove_clock_gating_check

Captures clock-gating checks.

SYNTAX

```
string remove_clock_gating_check
[-setup]
[-hold]
   [-rise]
   [-fall]
   [-high | -low]
   [object_list]

list object_list
```

ARGUMENTS

-setup

Indicates the removal of the clock-gating constraint on the setup time only. If you do not specify either the **-setup** or **-hold** option, both setup and hold constraints are removed.

-hold

Indicates the removal of the clock-gating constraint on the hold time only. If you do not specify either the **-setup** or **-hold** option, both setup and hold constraints are removed.

-rise

Indicates the removal of the clock-gating constaint on the rising delays only. If you do not specify either the **-rise** or **-fall** option, constraints on both rising and falling delays are removed.

-fall

Indicates the removal of the clock-gating constaint on the falling delays only. If you do not specify either the **-rise** nor **-fall** option, constraints on both rising and falling delays are removed.

-high

Remove the high specification from the obejct list, previously set up by set_clock_gating_check command. This option has to be either high or low..

-low

Remove the low specification from the obejct list, previously set up by set_clock_gating_check command. This option has to be either high or low.

object_list

Specifies a list of objects in the current design for which to remove the clock gating check. The objects can be clocks, ports, pins, or cells. If you specify a cell, all input pins of that cell are affected. If you do not specify any objects, the clock-gating check is removed from the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_clock_gating_check** command removes clock gating checks for design objects set by **set_clock_gating_check**.

EXAMPLES

The following example removes the setup requirement (for rising and falling delays) on all gates in the clock network involved with clock CK1 path.

```
ptc_shell> remove_clock_gating_check -setup [get_clocks CK1]
```

The following example removes the hold requirement on the rising delay of gate and1.

```
ptc_shell> remove_clock_gating_check -hold -rise [get_cells and1]
```

An alternative way to remove information set by **set_clock_gating_check** is to use the **reset_design** command.

SEE ALSO

set_clock_gating_check(2)
set_disable_clock_gating_check(2)
remove_disable_clock_gating_check(2)
current_design(2)
reset_design(2)

remove_clock_groups

Removes specific exclusive or asynchronous clock groups from the current design.

SYNTAX

```
Boolean remove_clock_groups
   -physically_exclusive | -exclusive | -asynchronous
   -name name_list | -all

list name_list
```

ARGUMENTS

-physically_exclusive

Specifies that groups set for physically exclusive clocks are to be removed. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-logically_exclusive

Specifies that groups set for logically exclusive clocks are to be removed. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-asynchronous

Specifies that groups set for asynchronous clocks are to be removed. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-name *name_list*

Specifies a list of clock groups to be removed, which matches the groups in the given names. You should use the **set_clock_groups** command to predefine these names. Substitute the list you want for *name_list*. The **-name** and **-all** options are mutually exclusive.

-all

Specifies to remove all groups set for exclusive or asynchronous clocks in the current design. The **-name** and **-all** options are mutually exclusive.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes specific exclusive or asynchronous clock groups from the current design. These clock groups are specified by the name_list from the current design. You must specify either **-exclusive** or **-asynchronous**. You must specify either *name_list* or **-all**.

To find the names for existing groups, use the **report_clock** command with the **-groups** option.

EXAMPLES

The following example removes exclusive clock groups named *mux*.

```
ptc_shell> remove_clock -exclusive -name mux
```

The following example removes all asynchronous clock groups from the current design.

```
ptc_shell> remove_clock -asynchronous -all
```

SEE ALSO

report_clock(2)
set_clock_groups(2)

remove_clock_jitter

Removes clock jitter information previously set by the **set_clock_jitter** command.

SYNTAX

```
status remove_clock_jitter
   -clock clock_list
```

ARGUMENTS

-clock *clock_list*

Specifies a list of clocks from which to remove the clock jitter.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command removes the cycle-to-cycle jitter or the duty-cycle jitter of the primary master clock that was previously set by the **set_clock_jitter** command.

EXAMPLES

The following example removes the cycle-to-cycle jitter of 0.5 and duty-cycle jitter of 0.7 specified on the primary master clock mclk

```
pt_shell> set_clock_jitter -clock [get_clocks mclk] -cycle 0.5 -duty_cycle 0.7
pt_shell> remove_clock_jitter -clock [get_clocks mclk]
```

SEE ALSO

set_clock_jitter(2)

remove_clock_latency

Removes clock latency information from specified objects.

SYNTAX

```
string remove_clock_latency [-source]
   [-clock clock_list]
   object_list

list    clock_list
list object_list
```

ARGUMENTS

-source

Specifies that clock source latency should be removed.

-clock *clock_list*

Removes any network latency defined on the pin/port objects in object_list which refers the clocks in clock_list from the design. If the **-clock** option is supplied when object_list refers to clock objects, a warning is issued that the option is not relevant in this case and execution of the command proceeds as if -clock was not given. This option does not remove a more general latency setting without any specific clock.

object_list

Provides a list of clocks, ports, or pins.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes clock latency information from specified objects. It removes the user-specified clock network or source latency information from specified objects. If a dynamic component of clock source latency has

been specified this will also be removed. Clock Network latency is the time it takes for a clock signal on the design to propagate from the clock definition point to a register clock pin. Clock Source latency (also called insertion delay) is the time it takes for a clock signal to propagate from its actual ideal waveform origin point to the clock definition point in the design. Clock network and source latency information is set on objects using the **set_clock_latency** command. See the **set_clock_latency** man page for more details.

To report clock network and source latency information, use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example removes clock latency information from the clock named **CLK1**.

```
ptc_shell> remove_clock_latency [get_clocks CLK1]
```

The following example removes clock source latency information from the clock named **CLK1**.

```
ptc_shell> remove_clock_latency -source \
[get_clocks CLK1]
```

SEE ALSO

create_clock(2)
remove_clock(2)
remove_clock_uncertainty(2)
report_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)

remove_clock_map

Removes custom clock mapping for block-to-top or SDC-to-SDC comparison.

SYNTAX

```
string remove_clock_map
   -clocks1 clocks1_list
   -clocks2 clocks2_list]
   [-scenario1 first_scenario1]
   [-scenario2 second_scenario2]
   [-block_design block_design]
   [-cells cell_list]
```

Data Types

```
   clocks1_list   list
   clocks2_list   list
   cell_list      list
```

ARGUMENTS

-clocks1 ***clocks1_list***

A list of source clocks of the clock mapping in *scenario1*. This list must be the same length as the *clocks2_list*.

-clocks2 ***clocks2_list***

A list of destination clocks of the clock mapping in *scenario2*. This list must be the same length as the *clocks1_list*.

-scenario1 ***first_scenario1***

The first (source) scenario of the clock mapping. In block-to-top comparison, this must be the top scenario.

-scenario2 ***second_scenario2***

The second (destination) scenario of the clock mapping. In block-to-top comparison, this must be the block scenario.

-block_design *block_design*

The block design that the clock mapping is removed from in block-to-top comparison. The clock mapping is removed from all instances of the *design*. This option is ignored if the *-cells* option is specified.

-cells *cell_list*

A list of cell instances to remove the clock mapping from. This option is valid only for block-to-top clock mapping.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes custom clock mapping for block-to-top or SDC-to-SDC comparison. For the clocks whose mapping are removed by this command, the default waveform based comparison is used to match the clocks.

EXAMPLES

The following example removes the mapping from the top_Clk in top_scenario of top design to the blk_Clk of blk_scenario in block design.

```
ptc_shell> current_design top
ptc_shell> remove_clock_map -scenario1 top_scenario -scenario2 blk_scenario
-clocks1 {top_Clk} -clocks2 {blk_Clk} -block_design block
```

SEE ALSO

set_clock_map(2)
report_clock(2)

remove_clock_transition

Removes clock transition time information.

SYNTAX

```
string remove_clock_transition clock_list

list clock_list
```

ARGUMENTS

clock_list

Specifies a list of clocks for which to remove clock transition time.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes clock transition information specified by the **set_clock_transition** command. To list all the set clock transition values, use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example removes clock transition information from all clocks in the current design.

ptc_shell> **remove_clock_transition [all_clocks]**

SEE ALSO

all_clocks(2)
report_clock(2)
set_clock_transition(2)
create_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)

remove_clock_uncertainty

Removes clock uncertainty information previously set by the **set_clock_uncertainty** command.

SYNTAX

```
string remove_clock_uncertainty
   [object_list |
    -from from_clock
      | -rise_from rise_from_clock
      | -fall_from fall_from_clock
    -to to_clock
      | -rise_to rise_to_clock
      | -fall_to fall_to_clock]
   [-rise]
   [-fall]
   [-setup]
   [-hold]
   [object_list]

list from_clock
list rise_from_clock
list fall_from_clock
list rise_to_clock
list fall_to_clock
list to_clock
list object_list
```

ARGUMENTS

-from *from_clock* -to *to_clock*

These two options specify the source and destination clocks for interclock uncertainty. You must specify either the pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to**, or *object_list*; you cannot specify both.

-rise_from *rise_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to rising edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options. Use **-rise_from** instead of the obsolete option **-from_edge rise**.

-fall_from *fall_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to falling edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options. Use **-fall_from** instead of the obsolete option **-from_edge fall**.

-rise_to ***rise_to_clock***

Same as the **-to** option, but indicates that *uncertainty* applies only to rising edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options. Use **-rise_to** instead of the obsolete option **-to_edge rise**.

-fall_to ***fall_to_clock***

Same as the **-to** option, but indicates that *uncertainty* applies only to falling edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options. Use **-fall_to** instead of the obsolete option **-to_edge fall**.

object_list

Specifies a list of clocks, ports or pins from which uncertainty information is to be removed. You can use either the pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to** options or the *object_list* option, but you cannot specify both; they are mutually exclusive.

-rise

Specifies that uncertainty is to be removed for only the rising clock edge. By default, uncertainty is removed for both rising and falling clock edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-rise_to** instead.

-fall

Specifies that uncertainty is to be removed for only the falling clock edge. By default, uncertainty is removed for both rising and falling clock edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-fall_to** instead.

-setup

Specifies that only setup check uncertainty is to be removed. By default, both setup and hold check uncertainties are removed.

-hold

Specifies that only hold check uncertainty is to be removed. By default, both setup and hold check uncertainties are removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes clock uncertainty information previously set by the **set_clock_uncertainty** command from clocks, ports, pins, or between specified clocks. To display clock uncertainty information, use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example removes uncertainty information from a clock named *CLK* and a pin named *clk_buf/Z*.

```
ptc_shell> remove_clock_uncertainty [get_clocks CLK]
ptc_shell> remove_clock_uncertainty [get_pins clk_buf/Z]
```

The following example removes interclock uncertainties between the *PHI1* and *PHI2* clock domains.

```
ptc_shell> remove_clock_uncertainty -from PHI1 -to PHI1
ptc_shell> remove_clock_uncertainty -from PHI2 -to PHI2
ptc_shell> remove_clock_uncertainty -from PHI1 -to PHI2
ptc_shell> remove_clock_uncertainty -from PHI2 -to PHI1
```

SEE ALSO

all_clocks(2)
create_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)

remove_data_check

Removes specified data-to-data checks previously set by the **set_data_check** command.

SYNTAX

```
string remove_data_check
   {-from from_object
      | -rise_from from_object
      | -fall_from from_object}
   {-to to_object
      | -rise_to to_object
      | -fall_to to_object}
   [-setup | -hold]
   [-clock clock_object]
```

Data Types

```
   from_object       object
   to_object         object
   clock_object      object
```

ARGUMENTS

-from *from_object*

Specifies a pin or port in the current design as the related pin of the data-to-data check is removed. Both rising and falling checks are removed. You must specify one of the *-from*, *-rise_from*, or *-fall_from* options.

-rise_from *from_object*

Similar to the *-from* option, but applies only to rising delays at the related pin. You must specify one of the *-from*, *-rise_from*, or *-fall_from* options.

-fall_from *from_object*

Similar to the *-from* option, but applies only to falling delays at the related pin. You must specify one of the *-from*, *-rise_from*, or *-fall_from* options.

-to *to_object*

Specifies a pin or port in the current design as the constrained pin of the data-to-data check is created. Both rising and falling checks are removed. You must specify one of the *-to*, *-rise_to*, or *-fall_to* options.

-rise_to ***to_object***

Similar to the *-to* option, but applies only to rising delays at the constrained pin. You must specify one of the *-to*, *-rise_to*, or *-fall_to* options.

-fall_to ***to_object***

Similar to the *-to* option, but applies only to falling delays at the constrained pin. You must specify one of the *-to*, *-rise_to*, or *-fall_to* options.

-setup

Indicates that only the setup data check is removed. If neither the *-setup* or *-hold* option is specified, both setup and hold checks are removed.

-hold

Indicates that only the hold data check is removed. If neither the *-setup* or *-hold* is specified, both setup and hold checks are removed.

-clock ***clock_object***

Indicates that the data check for the specified clock at the related pin is removed. This option applies only if a previous **set_data_check** command used the *-clock* option to specify the same clock; otherwise, this option is ignored.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_data_check** command specifies data-to-data checks are removed between the from object and to object for the specified options. These checks were previously set using the **set_data_check** command.

EXAMPLES

The following example removes a data-to-data check from and1/B to and1/A with respect to the rising edge of signals at and1/B. Both setup and hold checks are removed.

```
pt_shell> remove_data_check -rise_from and1/B -to and1/A
```

The following example removes setup data-to-data check between and1/B to and1/A with respect to the rising edge of signals at and1/B, coming from startpoints triggered with clock CK1, falling edge of signals at

and1/A.

```
pt_shell> remove_data_check -rise_from and1/B -fall_to and1/A -setup -clock [get_clock CK1]
```

SEE ALSO

set_data_check(2)

remove_design

Removes one or more designs from memory.

SYNTAX

```
string remove_design -all -hierarchy designs

list designs
```

ARGUMENTS

-all

Indicates that all designs are to be removed. **-all**, **-hierarchy**, and *designs* are mutually exclusive; you can specify only one.

-hierarchy

Indicates that all designs in the hierarchy of the current design are to be removed, including the current design. **-all**, **-hierarchy**, and *designs* are mutually exclusive; you can specify only one.

designs

Specifies a list of designs to remove. **-all**, **-hierarchy**, and *designs* are mutually exclusive; you can specify only one.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_design** command removes designs from ptc_shell. You must specify either **-all**, **-hierarchy** or *designs*, but not more than one of those. **remove_design** does not remove libraries; use the **remove_lib** command for that purpose.

Note that you cannot combine the **-hierarchy** option with a list of designs. **-hierarchy** is restricted to the current design. If the current design is not linked, it is automatically linked. Then, all designs in the

hierarchy of the current design are removed, including the current design itself.

EXAMPLES

The following example removes the design 'foo' and 'foo1'.

```
ptc_shell> remove_design {d1 d2}
Removing design 'd1'...
Removing design 'd2'...
1
```

The following example removes all designs in memory.

```
ptc_shell> remove_design -all
Removing design 'd3'...
1
```

The following example removes all designs in the hierarchy of the current design. In this example, the design was already linked.

```
ptc_shell> remove_design -hierarchy
Removing design 'TOP'...
Removing design 'eBlock'...
Removing design 'iBlock'...
Removing design 'mBlock'...
1
```

SEE ALSO

current_design(2)
list_designs(2)
remove_lib(2)

remove_disable_clock_gating_check

Restores clock gating checks previously disabled by **set_disable_clock_gating_check**, for specified cells and pins.

SYNTAX

```
string remove_disable_clock_gating_check object_list

list object_list
```

ARGUMENTS

object_list

Specifies a list of cells and pins for whom previously-disabled clock gating checks are to be restored.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Restores timing clock gating checks previously disabled with the **set_disable_clock_gating_check** command.

EXAMPLES

The following example restores disabled clock gating checks for the object an1.

ptc_shell> **remove_disable_clock_gating_check an1/A**

The following example restores all disabled gating checks on the cell U1/or2.

```
ptc_shell> remove_disable_clock_gating_check U1/or2
```

SEE ALSO

set_disable_clock_gating_check(2)

remove_disable_timing

Enables the previously disabled timing arcs.

SYNTAX

```
string remove_disable_timing
   [-from from_pin_name]
   [-to to_pin_name]
   object_list

string from_pin_name
string to_pin_name
list object_list
```

ARGUMENTS

-from ***from_pin_name***

-to ***to_pin_name***

Specifies that only the arcs between these two pins on the specified cell or library cell be disabled.

object_list

Specifies a list of cells, pins, library cells, library cell pins, or ports. The *object_list* must contain only cells or library cells (if **-from** and **-to** are specified).

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Restore timing arcs previously disabled with the **set_disable_timing** command. When **-from** and **-to** pins are specified, all arcs between these two pins on the cell or library cell are restored.

To list the timing arcs for a library cell, use **report_lib -timing_arcs**.

EXAMPLES

The following example restores disabled setup and hold arcs between pins CLK and TE on library cell SFF1.

ptc_shell> **remove_disable_timing -from CLK -to TE tech_lib/SFF1**

The following example restores all disabled cell arcs from or to pin A on cell U1/U2.

ptc_shell> **remove_disable_timing U1/U2/A**

SEE ALSO

report_lib(2)
set_disable_timing(2)
report_lib(2)

remove_drive_resistance

Removes drive resistance for input or inout ports.

SYNTAX

```
string remove_drive_resistance
   port_list
```

Data Types

```
   port_list     list
```

ARGUMENTS

port_list

Specifies a list of input or inout port names of the current design from which the drive values are to be removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_drive_resistance** command removes the drive resistance value from one or more input or inout ports. This is equivalent to using **set_drive_resistance 0**. In fact, that is how this command works. So, if output ports are passed into **remove_drive_resistance**, a DES-003 message is issued, indicating that **set_drive_resistance** could not be performed on those ports.

EXAMPLES

The following example shows the difference in the output report after using the **remove_drive_resistance** command.

```
ptc_shell> set_drive_resistance 5 [get_ports {A B C}]
1
ptc_shell> report_port -drive {A B C}

****************************************
Report : port
        -drive
Design : counter
****************************************


                Resistance          Transition
Input Port    Rise      Fall      Rise      Fall
------------------------------------------------------------
A             5.00      5.00      --        --
B             5.00      5.00      --        --
C             5.00      5.00      --        --

1
ptc_shell> remove_drive_resistance A
1
ptc_shell> report_port -drive {A B C}

****************************************
Report : port
        -drive
Design : counter
****************************************


                Resistance          Transition
Input Port    Rise      Fall      Rise      Fall
------------------------------------------------------------
A              --        --       --        --
B             5.00      5.00      --        --
C             5.00      5.00      --        --

1
```

SEE ALSO

```
report_port(2)
set_capacitance(2)
set_drive_resistance(2)
```

remove_driving_cell

Removes port driving cell information.

SYNTAX

```
string remove_driving_cell
   [-rise]
   [-fall]
   [-min]
   [-max]
   [-clock clock_name]
   [-clock_fall]
   port_list
```

Data Types

```
   clock_name     string
   port_list      list
```

ARGUMENTS

-rise

Removes rise driving cell information.

-fall

Removes fall driving cell information.

-min

Removes min driving cell information.

-max

Removes max driving cell information.

-clock *clock_name*

Removes the driving cell set relative to the specified clock.

-clock_fall

Removes the driving cell relative to the falling edge of the clock. The default is the rising edge.

port_list

Provides a list of input or output ports.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes driving cell information from the specified ports. The driving cell information is specified with the **set_driving_cell** command. The default is to remove all **-rise** and **-fall** information.

To see port drive information, use the **report_port -drive** command.

EXAMPLE

The following example removes all driving cell information for port IN2.

```
ptc_shell> remove_driving_cell IN2
```

SEE ALSO

report_port(2)
set_driving_cell(2)

remove_fanout_load

Removes fanout load information from output ports in the current design.

SYNTAX

```
string remove_fanout_load
   port_list
```

Data Types

```
   port_list         list
```

ARGUMENTS

port_list

Specifies a list of output ports.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes fanout load information specified by the **set_fanout_load** command. Fanout load for a net is the sum of the *fanout_load* attributes for all of the input pins and output ports connected to the net. Output pins can have maximum fanout limits, specified in the library or with the **set_max_fanout** command. The *fanout_load* is used when checking max_fanout design values.

EXAMPLES

The following example removes the fanout load on ports matching "OUT*".

```
ptc_shell> remove_fanout_load "OUT*"
```

SEE ALSO

set_fanout_load(2)
set_max_fanout(2)

remove_from_collection

Removes objects from a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection remove_from_collection
[-intersect]
   base_collection

x1collection base_collection
list  object_spec
```

ARGUMENTS

-intersect

Removes objects from collection1 not found in object_spec. Without this option, removes objects from collection1 that are found in object_spec.

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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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 constraint consistency gets all input ports except "CLOCK".

```
ptc_shell> set cPorts [remove_from_collection [all_inputs] CLOCK]
{"in1", "in2"}
```

SEE ALSO

add_to_collection(2)
collections(2)
query_objects(2)

remove_generated_clock

Removes generated clock objects from the current design.

SYNTAX

```
Boolean remove_generated_clock
   [-all]
   clock_list
```

Data Types

```
   clock_list     list
```

ARGUMENTS

-all

Indicates that all generated clocks are to be removed.

clock_list

Specifies a list of names of generated clocks to be removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes from the current design the generated clocks previously created using the **create_generated_clock** command. All attributes set on generated clocks (for example, **false_paths, multicycle_paths**) are also removed.

To display information about clocks and generated clocks in the design, use the **report_clock** command.

EXAMPLES

The following example removes the generated clock GEN1 from the design.

```
ptc_shell> remove_generated_clock GEN1
```

The following example removes all generated clocks from the design.

```
ptc_shell> remove_generated_clock -all
```

SEE ALSO

```
create_generated_clock(2)
report_clock(2)
```

remove_ideal_latency

Removes ideal latency values from the specified objects.

SYNTAX

```
int remove_ideal_latency
   [-rise]
   [-fall]
   [-min]
   [-max]
   object_list
```

Data Types

```
   object_list     list
```

ARGUMENTS

-rise

Specifies that only rise ideal latency should be removed. By default, both rise and fall ideal latency are removed.

-fall

Specifies that only fall ideal latency should be removed. By default, both rise and fall ideal latency are removed.

-min

Specifies that only minimum ideal latency should be removed. By default, both minimum and maximum ideal latency are removed.

-max

Specifies that only maximum ideal latency should be removed. By default, both minimum and maximum ideal latency are removed.

object_list

Specifies a list of ports or pins from where to remove ideal latency.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes the user-specified ideal latency that was previously set by the **set_ideal_latency** command from specified objects in the **object_list** argument.

You can remove selected portions of ideal latency by specifying applicable *-rise*, *-fall*, *-min*, and *-max* options.

EXAMPLES

The following example removes ideal latency from the output pin named *Z* of cell instance *U1/U2/U3*.

```
ptc_shell> remove_ideal_latency {U1/U2/U3/Z}
```

The following example removes only rise ideal latency information from a port named *A*.

```
ptc_shell> remove_ideal_latency -rise {A}
```

SEE ALSO

```
remove_ideal_network(2)
remove_ideal_transition(2)
report_timing(2)
set_ideal_network(2)
set_ideal_latency(2)
```

remove_ideal_network

Removes sources of ideal networks in the current design. Cells and nets in the transitive fanout of the specified objects are no longer treated as ideal.

SYNTAX

```
int remove_ideal_network
   object_list
```

Data Types

```
   object_list          list
```

ARGUMENTS

object_list

Specifies a list of ideal sources. The source objects can be ports or pins of leaf cells at any hierarchical level of the design. If nets are specified in the *object_list* option, all of the global driver ideal source pins from the nets that were set with the *-no_propagate* option are removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes the ideal network property from a set of ports or pins in the design that were previously marked as ideal sources using the **set_ideal_network** command. You specify only the source of the network. The ideal network is automatically updated by constraint consistency. The criteria for propagating the ideal property are detailed in the **set_ideal_network** command man page.

All ideal networks are removed using the **reset_design** command.

EXAMPLES

The following example sets up an ideal network on objects in the CLOCK_GEN design and then removes part of the network.

```
ptc_shell> current_design CLOCK_GEN
ptc_shell> set_ideal_network {port1 port2}
ptc_shell> remove_ideal_network port2
```

The following example creates an ideal network and then removes part of the ideal network.

```
ptc_shell> current_design {TOP}
ptc_shell> set_ideal_network {IN1 IN2}
ptc_shell> remove_ideal_network {IN1}
```

SEE ALSO

```
remove_ideal_latency(2)
remove_ideal_transition(2)
reset_design(2)
set_ideal_network(2)
```

remove_ideal_transition

Removes ideal transition values from the specified objects.

SYNTAX

```
int remove_ideal_transition
   [-rise]
   [-fall]
   [-min]
   [-max]
   object_list
```

Data Types

```
   object_list      list
```

ARGUMENTS

-rise

Specifies that only rise ideal transition should be removed. By default, both rise and fall ideal transitions are removed.

-fall

Specifies that only fall ideal transition should be removed. By default, both rise and fall ideal transitions are removed.

-min

Specifies that only minimum ideal transition should be removed. By default, both minimum and maximum ideal transitions are removed.

-max

Specifies that only maximum ideal transition should be removed. By default, both minimum and maximum ideal transitions are removed.

object_list

Specifies a list of ports or pins from which to remove ideal transition.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes the user-specified ideal transition that was previously set by the **set_ideal_transition** command from specified objects in the *object_list* option.

You can remove selected portions of ideal transition by specifying applicable *-rise*, *-fall*, *-min*, and *-max* options.

EXAMPLES

The following example removes ideal transition from the output pin named *Z* of cell instance *U1/U2/U3*.

```
ptc_shell> remove_ideal_transition {U1/U2/U3/Z}
```

The following example removes only rise ideal transition information from a port named *A*.

```
ptc_shell> remove_ideal_transition -rise {A}
```

SEE ALSO

```
remove_ideal_latency(2)
remove_ideal_network(2)
report_timing(2)
set_ideal_network(2)
set_ideal_transition(2)
```

remove_input_delay

Removes input delay information from ports or pins.

SYNTAX

```
string remove_input_delay
   [-clock clock_name]
   [-clock_fall]
   [-level_sensitive]
   [-rise]
   [-fall]
   [-max]
   [-min]
   port_pin_list
```

Data Types

```
   clock_name        list
   port_pin_list     list
```

ARGUMENTS

-clock *clock_name*

Relative clock; '' for no clock. Use this option to remove only input delay relative to one clock.

-clock_fall

Delay is relative to falling edge of clock.

-level_sensitive

Delay is from level-sensitive latch.

-rise

Removes rising input delay.

-fall

Removes falling input delay.

-max

Removes maximum input delay.

-min

Removes minimum input delay.

port_pin_list

Specifies a list of ports and pins.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes input delay information from ports or pins in the current design. Input delay is specified with the **set_input_delay** command. To show input delays on ports, use **report_port -input_delay**. The default is to remove all input delay information in the *port_pin_list* option.

EXAMPLES

The following example removes all input delay from ports IN*.

```
ptc_shell> remove_input_delay [get_ports IN*]
```

The following example removes input delay relative to CLK rise edge from port DATA[5].

```
ptc_shell> remove_input_delay -clock CLK { DATA[5] }
```

SEE ALSO

```
report_port(2)
set_input_delay(2)
set_output_delay(2)
remove_output_delay(2)
```

remove_lib

Removes one or more libraries from memory.

SYNTAX

```
string remove_lib
   -all
   libraries
```

Data Types

```
   libraries         list
```

ARGUMENTS

-all

Removes all libraries.

libraries

Provides a list of libraries to remove.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes a list of libraries. You must specify either the *libraries* or **-all** option. For a library to be removed, none of its library cells can be instantiated in any design, nor can any environment information (such as operating conditions or wire load models) be in use from the library. If this is the case, a message is issued and you must remove the design by first using the **remove_design** command, and then using the **remove_lib** command.

EXAMPLES

The following command removes all of the libraries that are read in.

```
ptc_shell> remove_lib -all
Removing library '/u/libraries/cmos1.db:cmos1'...
1
```

The following command removes a library that was part of a max/min pair created by the **set_min_library** command.

```
ptc_shell> remove_lib lib_ff
Removed max/min library relationship:
  Max: /u/libraries/lib_ss.db:lib_ss
  Min: /u/libraries/lib_ff.db:lib_ff
Removing library '/u/libraries/lib_ff.db:lib_ff'...
1
```

SEE ALSO

```
list_libraries(2)
list_designs(2)
remove_design(2)
set_min_library(2)
```

remove_linked_design

Removes one or more linked designs from memory but keep resolved references.

SYNTAX

```
string remove_design -all designs

list designs
```

ARGUMENTS

-all

Indicates that all linked designs are to be removed. **-all**, and *designs* are mutually exclusive; you can specify only one.

designs

Specifies a list of designs to remove. **-all**, and *designs* are mutually exclusive; you can specify only one.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_linked_design** command removes designs from ptc_shell. You must specify either **-all**, or *designs*, but not more than one of those. **remove_linked_design** does not remove verilog modules or libraries already loaded into memory; use **remove_design** and **remove_lib** command for those purposes.

EXAMPLES

The following example removes the linked designs 'foo' and 'foo1'.

```
ptc_shell> remove_linked_design foo*
Removing linked design 'foo'...
Removing linked design 'foo1'...
1
```

The following example removes all linked designs in memory.

```
ptc_shell> remove_linked_design -all
Removing design 'foo'...
Removing design 'foo1'...
Removing design 'foo2'...
1
```

Note that Verilog modules loaded into memory are not removed. So you can easily re-link them without explicit read_verilog.

```
ptc_shell> remove_linked_design foo
Removing design 'foo'...
ptc_shell> link_design foo
Design 'foo' was successfully linked.
1
```

SEE ALSO

current_design(2)
list_designs(2)
link_design(2)
remove_design(2)

remove_max_capacitance

Removes maximum capacitance limits from pins, ports, clocks, or designs.

SYNTAX

```
string remove_max_capacitance
   object_list
```

Data Types

```
   object_list          list
```

ARGUMENTS

object_list

Provides a list of pins, ports, clocks, or designs from which to remove maximum capacitance limits.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes maximum capacitance limits from pins, ports, clocks, or designs. A maximum capacitance limit is specified with the **set_max_capacitance** command.

EXAMPLES

The following example removes the maximum capacitance limit on ports "OUT*".

```
ptc_shell> remove_max_capacitance [get_ports "OUT*"]
```

The following example removes the maximum capacitance limit on the current design.

```
ptc_shell> remove_max_capacitance [current_design]
```

SEE ALSO

set_max_capacitance(2)

remove_max_fanout

Removes maximum fanout limits from ports or designs.

SYNTAX

```
string remove_max_fanout
   object_list
```

Data Types

```
   object_list       list
```

ARGUMENTS

object_list

Lists the ports or designs from which to remove maximum fanout limits.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes maximum fanout limits from ports or designs. A maximum fanout limit is specified with the **set_max_fanout** command.

EXAMPLES

The following example removes the maximum fanout limit on ports "OUT*".

```
ptc_shell> remove_max_fanout [get_ports "OUT*"]
```

The following example removes the maximum fanout limit on the current design.

```
ptc_shell> remove_max_fanout [current_design]
```

SEE ALSO

current_design(2)
get_ports(2)
set_max_fanout(2)
set_fanout_load(2)
set_max_capacitance(2)
set_max_transition(2)

remove_max_time_borrow

Removes time borrow limit for latches.

SYNTAX

```
string remove_max_time_borrow
   object_list
```

Data Types

```
   object_list        list
```

ARGUMENTS

object_list

Lists clocks, cells, data pins, or clock (enable) pins. If you specify a cell, all enable pins on that cell are affected.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes maximum time borrow limit on objects. Time borrowing limits are specified with the **set_max_time_borrow** command. When the limit is removed, full time borrowing is allowed on level-sensitive latches.

EXAMPLES

The following example removes the maximum time borrow limit on clock PHI1.

```
ptc_shell> remove_max_time_borrow [get_clocks PHI1]
```

The following example removes the maximum time borrow limit on all the pins of cells matching U1/latch*.

```
ptc_shell> remove_max_time_borrow [get_cells U1/latch*]
```

SEE ALSO

set_max_time_borrow(2)

remove_max_transition

Removes maximum transition limits from pins, ports, clocks or designs.

SYNTAX

```
string remove_max_transition
   object_list
```

Data Types

```
   object_list        list
```

ARGUMENTS

object_list

Lists the pins, ports, clocks, or designs from which to remove maximum transition limits.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes maximum transition limits from pins, ports, clocks or designs. A maximum transition limit is specified with the **set_max_transition** command.

EXAMPLES

The following example removes the maximum transition limit on ports "OUT*".

```
ptc_shell> remove_max_transition [get_ports "OUT*"]
```

The following example removes the maximum transition limit on the current design.

```
ptc_shell> remove_max_transition [current_design]
```

SEE ALSO

current_design(2)
set_max_transition(2)

remove_min_capacitance

Removes minimum capacitance limits from ports or designs.

SYNTAX

```
string remove_min_capacitance
   object_list
```

Data Types

```
   object_list       list
```

ARGUMENTS

object_list

Lists the ports or designs from which to remove minimum capacitance limits.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes minimum capacitance limits from ports or designs. A minimum capacitance limit is specified with the **set_min_capacitance** command.

EXAMPLES

The following example removes the minimum capacitance limit on ports OUT*.

```
pt_shell> remove_min_capacitance [get_ports "OUT*"]
```

The following example removes the minimum capacitance limit on the current design.

```
pt_shell> remove_min_capacitance [current_design]
```

SEE ALSO

current_design(2)
remove_max_capacitance(2)
set_min_capacitance(2)

remove_min_pulse_width

Removes a previously-specified minimum pulse width constraint from specified design objects.

SYNTAX

```
string remove_min_pulse_width
   [-low]
   [-high]
   [object_list]
```

Data Types

```
   object_list      list
```

ARGUMENTS

-low

Indicates that the minimum pulse width constraint is to be removed only for low clock signal levels. If you do not specify the *-low* or *-high* option, the constraint is removed for both low and high pulses of clock signals.

-high

Indicates that the minimum pulse width constraint is to be removed only for high clock signal levels. If you do not specify the *-low* or *-high* option, the constraint is removed for both low and high pulses of clock signals.

object_list

Specifies a list of clocks, cells, pins, or ports in the current design for which the minimum pulse width constraint is to be removed. If you specify a cell, all pins on that cell are affected. If you do not specify any objects, the minimum pulse width check is removed from all objects in the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_min_pulse_width** command removes the pulse width check previously specified by the **set_min_pulse_width** command for clock signals in clock trees or at sequential devices.

The **reset_design** command removes all user-specified attributes from a design, including those set by the **set_min_pulse_width** command.

EXAMPLES

The following example removes a minimum pulse width requirement previously set for clock CK1.

```
pt_shell> remove_min_pulse_width [get_clocks CK1]
```

The following example removes a minimum pulse width requirement previously set for pin U1/Z.

```
pt_shell> remove_min_pulse_width U1/Z
```

SEE ALSO

```
current_design(2)
reset_design(2)
set_min_pulse_width(2)
```

remove_operating_conditions

Removes operating conditions from current design, cells or ports.

SYNTAX

```
string remove_operating_conditions
   [-object_list objects]

list objects
```

ARGUMENTS

-object_list objects

Specifies the cells or ports to remove operating conditions from. The command undoes operating conditions set by **set_operating_conditions**. Without **-object_list** all the operating conditions are removed from the design. Both leaf cells and hierarchical blocks are accepted with **-object-list**.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes operating conditions settings from the current design, individual blocks, leaf cells, or ports.

EXAMPLES

This example removes all operating conditions from the current design. Both design-specific and cell-specific operating conditions are removed.

```
ptc_shell> remove_operating_conditions
```

SEE ALSO

set_operating_conditions(2)

remove_output_delay

Removes output delay from output ports or pins.

SYNTAX

```
string remove_output_delay
   [-clock clock_name]
   [-clock_fall]
   [-level_sensitive]
   [-rise]
   [-fall]
   [-max]
   [-min]
   port_pin_list

string clock_name
list port_pin_list
```

ARGUMENTS

-clock *clock_name*

Relative clock; {""} for input delay relative to no clock.

-clock_fall

Removes the delay relative to falling edge of clock. If you specify *clock_name* without **-clock_fall**, the delay relative to rising edge of the clock is removed.

-level_sensitive

Removes level-sensitive output delay.

-rise

Removes rising output delay.

-fall

Removes falling output delay.

-max

Removes maximum output delay.

-min

Removes minimum output delay.

port_pin_list

Specifies a list of ports and pins. Each element in the list is either a collection of ports or pins, or a pattern which matches ports or pins on the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes output delay values for objects in the current design. Set output delay with **set_output_delay**. By default, all output delays on each object in *port_pin_list* are removed. To restrict the removed output delay values, use **-clock**, **-clock_fall**, **-min**, **-max**, **-rise**, or **-fall**. To show output delays associated with ports, use **report_port -output_delay**.

EXAMPLES

The following example removes all output delay values from all output ports in the current design.

ptc_shell> **remove_output_delay [all_outputs]**

The following example removes output maximum rise delay values from OUT1, OUT3, and BIDIR12.

ptc_shell> **remove_output_delay -max -rise { OUT1 OUT3 BIDIR12 }**

The following example removes all output delays for port OUT1 relative to the falling edge of CLK2.

ptc_shell> **remove_output_delay -clock CLK2 -clock_fall OUT1**

The following example removes all output delays not relative to any clock for port OUT2.

ptc_shell> **remove_output_delay -clock {""} OUT2**

SEE ALSO

all_outputs(2)
collections(2)

report_port(2)
set_output_delay(2)
set_input_delay(2)

remove_path_group

Removes path group objects.

SYNTAX

```
string remove_path_group
   -all path_group_list
```

Data Types

```
   path_group_list      list
```

ARGUMENTS

-all

Removes all path groups in the current design.

path_group_list

Specifies path group names to remove. Each element in the list is either a collection of path groups or a pattern matching path group names.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes path groups in the current design. The **group_path** and **create_clock** commands create path groups. Path groups affect the cost function for optimization and influence the timing reports. If you remove a path group, any paths in that group are implicitly assigned to the default path group.

EXAMPLES

The following example removes all path groups in the design.

```
ptc_shell> remove_path_group -all
```

The following example removes path group "BUS1".

```
ptc_shell> remove_path_group BUS1
```

SEE ALSO

```
collections(2)
create_clock(2)
group_path(2)
```

remove_port_fanout_number

Removes fanout number information on ports.

SYNTAX

```
string remove_port_fanout_number
   port_list
```

Data Types

```
   port_list    list
```

ARGUMENTS

port_list

Specifies a list of ports. Each element in the list is either a collection of ports or a pattern that matches ports on the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes the **fanout_number** settings on ports.

EXAMPLES

This example removes the port **fanout_number** settings from ports OUT1*.

```
ptc_shell> remove_port_fanout_number [get_ports OUT1*]
```

SEE ALSO

set_port_fanout_number(2)

remove_propagated_clock

Removes a propagated clock specification.

SYNTAX

```
string remove_propagated_clock object_list

list object_list
```

ARGUMENTS

object_list

Lists clocks, ports, or pins.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes propagated clock specification from clocks, ports, or pins in the current design. The **set_propagated_clock** command specifies that delays be propagated through the clock network to determine latency at register clock pins. If this is not specified, ideal clocking is assumed. Ideal clocking means clock networks have a user specified latency (set by the **set_clock_latency** command) or zero latency, by default. Propagated clock latency is normally used for post layout, after final clock tree generation.

Ideal clock latency provides an estimate of the clock tree for prelayout. You can also use **set_clock_latency** to specify an ideal latency, which overrides the propagated clock specification for an object.

For information on propagated clock attributes on clocks, see the **report_clock** command man page.

EXAMPLES

The following example removes propagated clock specifications from all clocks in the design.

```
ptc_shell> remove_propagated_clock [all_clocks]
```

SEE ALSO

report_clock(2)
set_clock_latency(2)
set_propagated_clock(2)

remove_rule

Removes specified user-defined rules.

SYNTAX

```
Boolean remove_rule
   rule_list

list rule_list
```

ARGUMENTS

rule_list

The list of rules or rulesets to remove. If a ruleset is specified, all user-defined rules from that ruleset are removed, but the ruleset is not removed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes specified user-defined rules. It is an error to try to remove a built-in rule, or a ruleset containing built-in rules.

EXAMPLES

The following example removes the user-defined rule named "UDEF_0078".

```
ptc_shell> remove_rule UDEF_0078
```

SEE ALSO

analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
enable_rule(2)
get_rules(2)

remove_ruleset

Removes specified rulesets.

SYNTAX

```
Boolean remove_ruleset
   ruleset_list

list ruleset_list
```

ARGUMENTS

ruleset_list

The list of rulesets to remove.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes specified rulesets. Note that removing a ruleset does not remove the rules in that ruleset, as one rule may be in multiple rulesets. To remove all rules in a ruleset, use the **remove_rule** command instead.

EXAMPLES

The following example removes the ruleset named "my_set1".

```
ptc_shell> remove_ruleset my_set1
```

SEE ALSO

analyze_design(2)
create_rule(2)
create_ruleset(2)
disable_rule(2)
enable_rule(2)
get_rules(2)
remove_rule(2)

remove_scenario

Removes a list of scenarios in multi scenario analysis. The remove_mode command is a synonym for the remove_scenario command.

SYNTAX

```
remove_scenario
   scenario_list
   [-all]
```

ARGUMENTS

scenario_list

A list of unique strings used to identify each scenario.

-all

A shortcut to remove all the defined scenarios.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **remove_scenario** command removes the specified scenario from memory. To determine the current scenarios that exist, execute the following:

```
report_multi_scenario_design -scenarios.
1
```

EXAMPLES

In the following example, two scenarios named scen1 and scen2 are removed.

```
ptc_shell> remove_scenario {scen1 scen2}
1
```

SEE ALSO

create_scenario(2)
report_multi_scenario_design(2)

remove_sense

Removes sense information defined on pins or cell timing arcs.

SYNTAX

```
string remove_sense
   [-type type]
   [-clocks clocks_object_list]
   [-all]
   object_list
```

Data Types

```
   clocks_object_list         list
   object_list                list
```

ARGUMENTS

-type *type*

Specifies whether the type of sense being removed refers to clock networks or data networks. The possible values for the *type* variable are: 'clock', 'data'. Note that 'clock' is assumed to be the default if *-type* option is not given.

-clocks *clocks_object_list*

Optionally specifies a list of clock objects to be associated with the given pin objects in the *object_list* variable. If the *-clocks* option is specified, only the unateness specified for that particular clock domain is removed. Otherwise, unateness information for all clocks passing through the given pin objects is removed. The *-clocks* option can only remove clock sense predefined by the **set_clock_sense** *-clock* variable. It does not remove the default clock sense setting for this given pin.

-all

Removes all unateness information in current design.

object_list

Lists pins or cell timing arcs with predefined unateness to remove.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes the unateness information which you predefined. To set the unateness information, use the **set_sense** command.

EXAMPLES

The following example removes positive unateness defined on a pin named **XOR/Z** with respect to clock **CLK1**, but does not remove the negative unateness.

```
pt_shell> set_sense -positive -clocks [get_clocks CLK1] XOR/Z
pt_shell> set_sense -negative XOR/Z
pt_shell> remove_sense -clocks [get_clocks CLK1] XOR/Z
```

The following example removes all unateness information in the current design.

```
pt_shell> remove_sense -all
```

SEE ALSO

set_sense(2)

remove_waiver

Remove violation waivers satisfying the given requirements.

SYNTAX

```
Boolean remove_waiver
   [-names waiver_name_list]
   [-rules rule_name_list]
   [-design design]
   [-scenarios scenario_name_list]
```

Data Types

```
   waiver_name_list      list
   rule_name_list        list
   design                string
   scenario_name_list    list
```

ARGUMENTS

-names *waiver_name_list*

Remove waivers with names in the given list.

-rules *rule_name_list*

Remove waivers of the givens rules.

-design *design*

Remove the waivers for the given design only.

-scenarios *scenario_name_list*

Remove waivers active in the given scenarios (waivers for scenario-dependent rules only). This option should be used in conjuction of the "-design" option.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Remove waivers with the given names, for the given rules, and active in the given scenarios of the given design. If no *-names* option given, all waiver names are included; if no *-rules* option is given, waivers for all rules are included; if no *-scenarios* option given, waivers active in any scenario, including waivers for scenario-independent rules, are included. If none of the *-names*, *-rules*, *-design*, or *-scenarios* options are given, all existing waivers will be removed.

Note that multiple waivers can be applied that affect same instances. For example, using the *create_waiver -cells* option, instance based waivers can be set on a hierarchical cell and also on a particular instance inside the hierarchical cell. In such cases, all the waivers affecting the instance should be removed before the waiver is completely gone.

EXAMPLES

The following example removes a waiver named my_waiver1:

```
ptc_shell> remove_waiver -names my_waiver1
```

SEE ALSO

```
analyze_design(2)
create_waiver(2)
report_waiver(2)
write_waiver(2)
```

rename

Renames or deletes a command.

SYNTAX

```
string rename
   old_name
   new_name
```

ARGUMENTS

old_name

Specifies the current name of the command.

new_name

Specifies the new name of the command.

DESCRIPTION

Renames the *old_name* command so that it is now called *new_name*. If *new_name* is an empty string, then *old_name* is deleted. The *old_name* and *new_name* arguments may include namespace qualifiers (names of containing namespaces). If a command is renamed into a different namespace, future invocations of it will execute in the new namespace. The **rename** command returns an empty string as result.

Note that the **rename** command cannot be used on permanent procedures. Depending on the application, it can be used on all basic builtin commands. In some cases, the application will allow all commands to be renamed.

WARNING: **rename** can have serious consequences if not used correctly. When using **rename** on anything other than a user-defined Tcl procedure, you will be warned. The **rename** command is intended as a means to wrap other commands: that is, the command is replaced by a Tcl procedure which calls the

original. Parts of the application are written as Tcl procedures, and these procedures can use any command. Commands like **puts**, **echo**, **open**, **close**, **source** and many others are often used within the application. Use **rename** with extreme care and at your own risk. Consider using **alias**, Tcl procedures, or a private namespace before using **rename**.

EXAMPLES

This example renames my_proc to my_proc2:

```
prompt> proc my_proc {} {echo "Hello"}
prompt> rename my_proc my_proc2
prompt> my_proc2
Hello
prompt> my_proc
Error: unknown command 'my_proc' (CMD-005).
```

SEE ALSO

define_proc_attributes(2)

report_analysis_coverage

Generates a report about coverage of timing checks.

SYNTAX

```
string report_analysis_coverage
   [-scenarios scenario_list]
   [-status_details status_list]
   [-style style]
   [-check_type check_type_list]
   [-exclude_untested untested_reason_list]
   [-sort_by sort_method]
   [-nosplit]
   [pin_port_list]
```

Data Types

```
   scenario_list            list
   status_list              list
   style                    string
   check_type_list          list
   untested_reason_list     list
   sort_method              string
   pin_port_list            list
```

ARGUMENTS

-scenarios *scenario_list*

A list of scenarios to include in the output. If not specified, all scenarios are included.

-status_details *status_list*

Specifies a space-separated list of status names about which to show detail in the report. Allowed values are *untested* and *tested*. By default, only summary information is shown. Use this option to see information about individual timing checks.

-style *style*

Specifies the format of the details section of the report. Allowed values are *normal* and *verbose*. If the *verbose* style is specified, the details section includes untested reason for each check in each scenario.

-check_type *check_type_list*

Specifies a list of check types to include in the report. Allowed values are *setup*, *hold*, *recovery*, *removal*, *nochange*, *min_period*, *min_pulse_width*, *clock_separation*, *clock_gating_setup*, *clock_gating_hold*, *max_skew*, *out_setup*, and *out_hold*.

Note: The check types *out_setup* and *out_hold* refer to the output setup and output hold constraints generated by the **set_output_delay** command.

-exclude_untested *untested_reason_list*

Specifies a space-separated list of reasons to exclude an untested check in the report. A check classified as untested is excluded from the report for any of the reasons specified as values to this argument. Therefore, the coverage statistics are unaffected by these excluded constraints. Allowed values are as follows:

- *case_disabled*: Paths to this check are disabled because of a logic constant propagated through the design due to case analysis (example: set_case_analysis 0/1 port/pin).
- *constant_disabled*: Paths to this check are disabled because of a logic constant propagated through the design by a signal tied high or low (example: a pin or port tied to logic high or low).
- *mode_disabled*: A timing constraint is disabled because it requires a mode to be selected in mode analysis, and that mode is not selected.
- *user_disabled*: The timing check is explicitly disabled by you.
- *no_paths*: The timing check had no paths found to it and as a result there were no arrival times.
- *false_paths*: All paths were false to a constrained pin/port.
- *no_endpoint_clock*: The timing check has no destination clock signal to latch the data.
- *no_startpoint_clock*: The timing check has no clock that launches the data at a startpoint latch.
- *no_constrained_clock*: There is no constrained clock for skew or clock separation checks.
- *no_ref_clock*: There is no reference clock for skew or clock separation checks.
- *no_clock*: A minimum pulse width or period width check has no clock.
- *unknown*: The reason is not one of the reasons listed above.

-sort_by *sort_method*

Specifies the method of sorting for the output of the detailed list. Allowed values are *none*, *status*, *name*, and *check_type*. The default is *none* which does not perform any sorting on the output. This results in faster runtime for report creation. If you specify *status*, the output is sorted first by status (untested or tested), then by name, then by type of check. If you specify *name*, the output is sorted by pin/port name, then by status, then by check type. If you specify *check_type*, the output is sorted by type of check, then by status, then by pin/port name.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing tools to extract information from the report output.

pin_port_list

Specifies a list of pin/port names or a pin/port collection to be reported. If no pin_port_list is given then all pins/ports involved in checks in the design are reported.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Generates a report showing information about the coverage of timing checks in the current design, or current instance if it is defined. The default report is a summary of checks by type. For each type of check (for example, *setup*), the report shows the number and percentage of checks that are tested (constrained), and those that are untested; the report does not show check types for which there are no checks. The report does not show unconstrained output ports; that is, those that have no output delay or *max_delay* or *min_delay*; however, the report does show constrained output ports.

You can see more information about the individual timing checks using the **-status_details** option, which shows all checks with the corresponding status. Untested checks show information about why they are untested, if the reason can be determined.

By default, the coverage statistics include constraints that have been disabled. Constraints could be disabled because of constant propagation (for logic constants in the design and case analysis), modes that are not enabled, or explicit disabling of timing arcs by you. Such disabled constraints appear as untested in the coverage statistics. If you want to exclude some of these untested constraints from the coverage statistics, use the **-exclude_untested** option.

If the report is generated for more than one scenario, the summary section shows checks that are tested in at least one scenario separate from those that are untested in all scenarios. A typical use would be to define scenarios for all possible functional and test modes then run this report to see if any timing checks are not covered by those scenario constraints. If there are any such checks, more modes might be needed to ensure that the design is constrained under all possible conditions.

EXAMPLES

The following example shows the summary report.

```
ptc_shell> report_analysis_coverage
                                      Tested in         Untested in
Type of Check              Total      1+ scenario       all scenarios
-------------------------------------------------------------------------
clock_gating_hold           9400      4165 ( 44%)         5235 ( 56%)
clock_gating_setup          9400      4165 ( 44%)         5235 ( 56%)
hold                     1328927    575149 ( 43%)       753778 ( 57%)
min_period                   352       351 (100%)            1 (  0%)
min_pulse_width           870706    491034 ( 56%)       379672 ( 44%)
out_hold                      36        36 (100%)            0 (  0%)
out_setup                     36        36 (100%)            0 (  0%)
```

```
recovery                  315842    295658 ( 94%)         20184 (  6%)
removal                   315838    295654 ( 94%)         20184 (  6%)
setup                    1328927    575149 ( 43%)        753778 ( 57%)
-------------------------------------------------------------------------
All Checks               4179464   2241397 ( 54%)       1938067 ( 46%)
```

The detailed reports in multiscenario analysis let you know the modes where a particular check is tested. For example, you could run:

```
ptc_shell> report_analysis_coverage -check_type out_hold \
                   -style verbose -status_details tested

                                      Tested in          Untested in
Type of Check              Total      1+ scenario        all scenarios
-------------------------------------------------------------------
out_hold                      36        36 (100%)             0 (  0%)
-------------------------------------------------------------------
All Checks                    36        36 (100%)             0 (  0%)

. . .

Constrained        Related                                    Untested
Pin                Pin          Check Type         Scenarios  Reasons
-------------------------------------------------------------------
zOut1                           out_hold           default    x
                                                   FF_FUNC    x
                                                   FF_SCAN    t
zOut2                           out_hold           default    x
                                                   FF_FUNC    t
                                                   FF_SCAN    x
zOut3                           out_hold           default    x
                                                   FF_FUNC    t
                                                   FF_SCAN    x
. . .
```

In the Untested Reasons column, 't' stands for tested. As a consequence, you can deduct that zOut1 is being tested in scan mode, whereas zOut2 and zOut3 are being tested in functional mode.

SEE ALSO

analyze_design(2)
current_instance(2)

report_app_var

Shows the application variables.

SYNTAX

```
string report_app_var
   [-verbose]
   [-only_changed_vars]
   [pattern]
```

Data Types

```
   pattern      string
```

ARGUMENTS

-verbose

Shows detailed information.

-only_changed_vars

Reports only changed variables.

pattern

Reports on variables matching the pattern. The default is "*".

DESCRIPTION

The **report_app_var** command prints information about application variables matching the supplied pattern. By default, all descriptive information for the variable is printed, except for the help text.

If no variables match the pattern, an error is returned. Otherwise, this command returns the empty string.

If the **-verbose** option is used, then the command also prints the help text for the variable. This text is

printed after the variable name and all lines of the help text are prefixed with " #".

The Constraints column can take the following forms:

{val1 ...}

The valid values must belong to the displayed list.

val \<= a

The value must be less than or equal to "a".

val \>= b

The value must be greater than or equal to "b".

b \<= val \<= a

The value must be greater than or equal to "b", and less than or equal to "a".

EXAMPLES

The following are examples of the **report_app_var** command:

```
prompt> report_app_var sh*
Variable                 Value     Type    Default   Constraints
------------------------ --------- ------- --------- --------------------
sh_continue_on_error     false     bool    false
sh_script_stop_severity  none      string  none      {none W E}
 \.\.\.


prompt> report_app_var sh* -verbose
Variable                 Value     Type    Default    Constraints
------------------------ --------- ------- --------- --------------------
sh_continue_on_error     false     bool    false
   # Allows source to continue after an error
sh_script_stop_severity  none      string  none      {none W E}
   # Indicates the error message severity level which would cause
   # a script to stop executing before it completes
 \.\.\.
```

SEE ALSO

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

report_attribute

Reports the attributes on one or more objects.

SYNTAX

```
string report_attribute [-class class_name]
   [-nosplit]
   [-application]
   object_spec

string  class_name
list    object_spec
```

ARGUMENTS

-class *class_name*

If object_spec is a name, this is its class. Allowable values are design, cell, pin, port, net, lib, lib_cell, lib_pin, timing_arc, lib_timing_arc, clock, scenario, input_delay, output_delay, exception, exception_group, clock_group, clock_group_group, rule, ruleset, and rule_violation.

-nosplit

Does not split lines if column overflows.

-application

Lists application attributes.

object_spec

List of objects to report. Each element in the list is either a collection or a pattern combined with the *class_name* to find the objects.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Generates a report of attributes on the specified objects. The **-application** option should be used to report the application attributes for the objects.

Currently, constraint consistency does not supported user-defined attributes as PrimeTime does. The **-application** option is just to make constraint consistency to follow the same use model as PrimeTime.

For application attributes which are of the type *collection*, the name of the first object in the collection will be displayed.

EXAMPLES

This example shows how application attributes can be displayed.

```
ptc_shell> report_attribute -application [get_designs my_des]
```

****
Report : Attribute
Design : my_des
****
Design Object Type Attribute Name Value

----
my_des my_des string full_name my_des
my_des my_des string object_class design
my_des my_des float area 56.000000
my_des my_des string tree_type_max balanced_case
my_des my_des string tree_type_min balanced_case
my_des my_des float wire_load_min_block_size 0.000000
my_des my_des string wire_load_mode top
This example shows how to query available violation details for a given rule using report_attribute and utilize that information to further query the clock list of a corresponding rule violation. Then it retrieves the waveform attribute of the clock list.
ptc_shell> **report_attribute -application [get_rule CGR_0001]**
****
Report : Attribute
Design : CGR_0001
****
Design Object Type Attributes Value

CGR_0001 CGR_0001 string description Clocks which are generated from the same master clock cannot be asynchronous with each other.
CGR_0001 CGR_0001 string full_name
CGR_0001 CGR_0001 CGR_0001 boolean is_enabled true
CGR_0001 CGR_0001 string message {Clocks generated from the same master clock } { are declared as asynchronous.}
CGR_0001 CGR_0001 string object_class rule
CGR_0001 CGR_0001 string parameters {master_clock}
CGR_0001 CGR_0001 string severity error
CGR_0001 CGR_0001 string violation_details {clock1_list} {clock2_list}
1
ptc_shell> **set clock_list [get_violation_info -attribute clock1_list [get_rule_violation -of CGR_0001]]**
{"clk2"}
ptc_shell> **get_attribute $clock_list waveform**
{0.000000 4.000000}

SEE ALSO

collections(2)
get_attribute(2)
list_attributes(2)
get_violation_info(2)

report_case_analysis

Reports case analysis entries on ports and pins.

SYNTAX

```
string report_case_analysis
   [-all]
   [-nosplit]
```

ARGUMENTS

-all

Reports the pins upon which you have set case analysis values and reports the built-in constant pins of the design that are considered for start points of logic constant propagation. Logic constant propagation is performed by default from the constant pins of the design, unless the **disable_case_analysis** variable is set to true.

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, 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.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reports case analysis entries on ports and pins that are specified with the **set_case_analysis** command. The report lists all pins and ports on which case analysis is specified. The list does not report where the logic constant values are propagated.

EXAMPLES

The following example specifies that pins *U1/U2/A* and *U1/U3/CI* are set to a constant logic *1*.

```
   ptc_shell> set_case_analysis 1 {U1/U2/A U1/U3/CI}
   ptc_shell> report_case_analysis
****************************************
Report : case_analysis
Design : middle
Version: ...
Date   : Mon Jul 22 17:04:17 1996
****************************************

Pin name                                          User case analysis value
--------------------------------------------------------------------------
U1/U2/A                                           1
U1/U3/CI                                          1
```

SEE ALSO

remove_case_analysis(2)
set_case_analysis(2)

report_case_details

Report the propagation of user case values and logic constant values throughout the netlist. This command can show how case propagates to or from the specified pins/ports.

SYNTAX

```
string report_case_details
   [-sources]
   [-to]
   [-from]
   [-max_objects num]
   [-nosplit]
   [-backward_trace]
   [-text_only]
   pin_or_port_list
```

ARGUMENTS

-sources

Lists the pins/ports whose case values or logic constant values are propagated to each pin or port in the specified **pin_or_port_list**.

-to

Show the backward trace of case propagation to each pin or port specified in **pin_or_port_list**. The backward trace only expands to pins/ports whose case values are contributing to the case value on the destination pin/port. If the number of pins/ports involved in backward trace exceeds the number set in **-max_objects**, the trace will be truncated, and a message will be printed at the end of the trace indicating it is an incomplete trace.

-from

Show the forward trace of case propagation from each pin or port specified in **pin_or_port_list**. The forward trace only expands to pins/ports where the case value from the source is contributing to the case value on that pin/port. If the number of pins/ports involved in forward trace exceeds the number set in **-max_objects**, the trace will be truncated, and a message will be printed at the end of the trace indicating it is an incomplete trace.

-max_objects

Specify the maximum number of pins/ports allowed in a backward/forward case propagation trace report. The default value is 1000.

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, 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.

-backward_trace

This option will report conflict information in the fanin of set_case_analysis settings. When this option is used with the -to option, the pin or port list given to the -to option must have a set_case_analysis settings. When -backward_trace and -to options are given, the command will report any conflicts found backwards from the set_case_analysis locations given and any other set_case_analysis settings. If this option is used with -from <pin_or_port_list>, it will report on any set_case_analysis settings in the fanout of the given pins reporting conflicts.

-text_only

This option will prevent the schematic generation of the report requested. With this option only the text report is generated on the ptc_shell. This option has an effect only when the GUI is up. Without the GUI, the command will always generate a text report only.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Report the propagation of user case values and logic constant values throughout the netlist. This command can show how case propagates to or from the specified pins/ports.

When constraint consistency is running with the GUI interface this command will also generate the schematic showing the connections that are reported by the output of this command. If only the -sources option is used then no schematic will be generated.

EXAMPLES

The following example displays the sources (user set case and logic constant) of the case value on pin *y/Z*.

```
ptc_shell> report_case_details -sources y/Z
************************************
Report : case_details
Design : counter
Version: ...
Date   : Fri Feb 15 14:42:23 2008
```

```
**************************************
Properties                    Value     Pin/Port
------------------------------------------------------------------------------
from user case                1         y/Z


Case sources report:
Sources for pin/port y/Z
Properties                    Value     Pin/Port
------------------------------------------------------------------------------
user case                     1         L
user case                     1         P
user case                     1         T
```

The following example displays the fanin trace of the case value on pin *y/Z*.

```
   ptc_shell> report_case_details -to y/Z
**************************************
Report : case_details
Design : counter
Version: ...
Date   : Thu Feb 21 14:05:15 2008
**************************************
Properties         Value   Pin/Port
------------------------------------------------------------------------------
from user case     1       y/Z


Case fanin report:
Verbose Source Trace for pin/port y/Z:
Path number: 1
Path Ref #   Value  Properties           Pin/Port
------------------------------------------------------------------------------
             1      F()=A B & C &        y/Z
2            1                           y/A
3            1                           y/B
4            1                           y/C


Path number: 2
Path Ref #   Value  Properties           Pin/Port
------------------------------------------------------------------------------
             1                           y/A
             1      user case            P


Path number: 3
Path Ref #   Value  Properties           Pin/Port
------------------------------------------------------------------------------
             1                           y/B
             1      user case            T


Path number: 4
Path Ref #   Value  Properties           Pin/Port
------------------------------------------------------------------------------
             1                           y/C
             1      user case            L
```

The following example displays the fanout trace of the case value on port *CL*, with **-max_objects** set to

10.

```
ptc_shell> report_case_details -from -max_objects 10 CL
```

```
*************************************
Report : case_details
Design : counter
Version: ...
Date   : Thu Feb 21 14:09:50 2008
*************************************
Properties         Value   Pin/Port
-------------------------------------------------------------------------------
user case          1       CL


Case fanout report:
Verbose Forward Trace for pin/port CL:

Pin/Port ID  Value  Fanout Pin/Port IDs  Pin/Port
-------------------------------------------------------------------------------
0            1      1 8                 CL
1            1      3                   p/B
2            1      3                   p/A
3            1      4 5 6 7             p/Z
4            1                          a/D
5            1                          j/D
6            1                          g/D
7            1                          d/D
8            1                          o/A
9            0                          o/B
---- Number of fanout objects exceeds "-max_objects" ----
```

The following example displays the conflicting case values found with a backward trace from a case setting. *CL*, with **-max_objects** set to 10.

```
ptc_shell> report_case_details -to b6/A -backward_trace
*************************************
Report : case_details
Design : test
*************************************

set_case_analysis Sources Traced Backwards:

Source
ID    Value Pin/Port                File/Line
-------------------------------------------------------------------------------
a)    0     b2/A                    case.tcl, line 90
b)    1     b6/Z                    case.tcl, line 10

Fanout Report Backward Trace Values:

Pin/Port Implied    Fanout
ID       Values     Pin/Port IDs         Pin/Port
-------------------------------------------------------------------------------
0        0(a),1(b)  1,2                    b1/Z
1        0(a)                              b2/A
2        1(b)       3                      b6/A
3        1(b)                              b6/Z
```

SEE ALSO

report_case_analysis(2)
remove_case_analysis(2)
set_case_analysis(2)

report_cell

Reports cell information.

SYNTAX

```
string report_cell
   [-connections [-verbose]]
   [-significant_digits digits]
   [-nosplit]
   [cell_names]

list cell_names
```

ARGUMENTS

-connections

Displays the pins and the nets to which they connect.

-verbose

Displays verbose connection information. For each input pin, displays the net and the driver pins on that net. For each output pin, displays the net and the load pins on that net. Use this option in conjunction with **-connections** option.

-nosplit

Does not split lines if column overflows.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. Fixed-precision floating-point arithmetics is used for delay calculation; therefore, the actual precision might depend on the platform. For example, on SVR4 UNIX see FLT_DIG in limits.h. The upper bound on a meaningful value of the argument to **-significant_digits** is then FLT_DIG - ceil(log10(fabs(*any_reported_value*))).

cell_names

Lists cells to report. If you do not specify this option, all cells in the current instance are listed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Displays information and statistics about cells in the current instance or current design. If you set for the current instance, the report is generated for the design of that instance. Otherwise, the report is generated for the current design.

Some information, such as whether the cell is in an ideal network or not, is displayed after a timing update (as a result of manually executing an update_timing function or experiencing an implicit timing update as a result of executing other commands). This behavior is intended to help you to obtain information and statistics about cells without incurring the cost of a complete timing update.

EXAMPLES

The following example displays a report of the cells in the current design.

```
ptc_shell> report_cell
****************************************
Report : cell
Design : middle
Version: ...
Date   : Wed Mar  8 03:18:34 2006
****************************************

Attributes:
    b - black-box (unknown)
    h - hierarchical
    n - noncombinational
    u - contains unmapped logic
    A - abstracted timing model
    E - extracted timing model
    S - Stamp timing model
    Q - Quick timing model (QTM)
    I - ideal network

Cell                    Reference       Library             Area  Attributes
--------------------------------------------------------------------------------
i1                      inter                                6.00  h
low_i                   low                                  2.00  h
n0_i                    ND2             my_lib               1.00
o_reg2                  FD2             my_lib               9.00  n
--------------------------------------------------------------------------------
Total 4 cells                                               18.00

1
```

The following example gives a verbose report of the cell connections.

```
ptc_shell> report_cell -verbose -connections
****************************************
Report : cell
 -connections
 -verbose
Design : middle
Version: ...
Date   : Wed Mar  8 03:18:44 2006
****************************************


Connections for cell 'i1':
    Reference:          inter
    Hierarchical:       TRUE
    Area:               6

    Input Pins          Net             Net Driver Pins   Driver Pin Type
    ----------------    ------------    ----------------  ----------------
    a                   out_1           o_reg1/Q          Output Pin (FD2)
    b                   out_2           o_reg2/Q          Output Pin (FD2)
    c                   out_3           o_reg3/Q          Output Pin (FD2)
    d                   out_4           o_reg4/Q          Output Pin (FD2)

    Output Pins         Net             Net Load Pins     Load Pin Type
    ----------------    ------------    ----------------  ----------------
    z1                  i1t1            low_i/n1/A        Input Pin (NR4)
    z2                  i1t2            low_i/n1/B        Input Pin (NR4)
    z3                  i1t3            low_i/n1/C        Input Pin (NR4)

Connections for cell 'low_i':
    Reference:          low
    Hierarchical:       TRUE
    Area:               2

    Input Pins          Net             Net Driver Pins   Driver Pin Type
    ----------------    ------------    ----------------  ----------------
    a                   i1t1            i1/low/n1/Z       Output Pin (NR4)
    b                   i1t2            i1/low2/n1/Z      Output Pin (NR4)
    c                   i1t3            i1/low3/n1/Z      Output Pin (NR4)
    d                   in2             in2               Input Port

    Output Pins         Net             Net Load Pins     Load Pin Type
    ----------------    ------------    ----------------  ----------------
    z                   low             o_reg2/D          Input Pin (FD2)

Connections for cell 'n0_i':
    Reference:          ND2
    Library:            my_lib
    Area:               1

    Input Pins          Net             Net Driver Pins   Driver Pin Type
    ----------------    ------------    ----------------  ----------------
    A                   p_in            p_in              Input Port
    B                   p_in            p_in              Input Port

    Output Pins         Net             Net Load Pins     Load Pin Type
    ----------------    ------------    ----------------  ----------------
    Z                   n0              n1_i/B            Input Pin (ND2)

Connections for cell 'o_reg2':
    Reference:          FD2
    Library:            my_lib
    Area:               9
```

```
    Input Pins          Net            Net Driver Pins   Driver Pin Type
    ----------------    ------------   ----------------  ----------------
    D                   low            low_i/n1/Z        Output Pin (NR4)
    CP                  CLOCK          CLOCK             Input Port
    CD                  OPER           OPER              Input Port

    Output Pins         Net            Net Load Pins     Load Pin Type
    ----------------    ------------   ----------------  ----------------
    Q                   out_2          i1/low3/n1/B      Input Pin (NR4)
                                       out_2             Output Port
                                       i1/low/n1/B       Input Pin (NR4)
                                       i1/low2/n1/B      Input Pin (NR4)
    QN                  NET2QNX        i2/low3/n1/B      Input Pin (NR4)
                                       i2/low/n1/B       Input Pin (NR4)
                                       i2/low2/n1/B      Input Pin (NR4)

1
```

SEE ALSO

current_design(2)
current_instance(2)
report_design(2)
report_hierarchy(2)
report_net(2)
report_port(2)
report_reference(2)

report_clock

Reports clock-related information.

SYNTAX

```
string report_clock
   [-attributes]
   [-skew]
   [-groups]
   [-map]
   [-map_of instance_list]
   [-nosplit]
   [clock_names]
```

Data Types

```
   instance_list   list
   clock_names     list
```

ARGUMENTS

-attributes

Shows clock attributes and provides a list of all the clocks in the current design. The information for each clock includes source type, signal rise and fall times, and attributes. This report is shown by default.

-skew

Reports clock latency (source and network latency) and uncertainty information set on the design by the **set_clock_latency** and **set_clock_uncertainty** commands, respectively.

Clock network latency information includes rise latency and fall latency. Clock source latency information includes rise latency and fall latency for early and late arrivals. Clock uncertainty information includes intraclock setup or hold uncertainty and interclock setup or hold uncertainty. This option also reports any fixed clock transition set by using the **set_clock_transition** command. This option only reports active clocks.

-groups

Shows the current setting of clock groups, including the list of active clocks in the current analysis scope and grouping of exclusive clocks and asynchronous clocks set by using the **set_clock_groups** command.

-map

Shows the clock mappings for clocks in the current scenario.

-map_of ***instance_list***

Shows the clock mappings for clocks in the current scenario for the given instances specified in *instance_list*.

-nosplit

Specifies not to split lines if a column overflows. 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. This option prevents line-splitting and facilitates writing tools to extract information from the report output.

clock_names

Lists the clocks that must be reported. Substitute the list you want for *clock_names*.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reports clock-related information. Displays all clock-related information on a design. The report contents are controlled by the options used. If you do not specify any options, the command displays the attributes report.

When a clock is created, it defaults to being active and is added to the active clocks list. The new clock is also added to the other clocks list if some clocks are defined to be exclusive or asynchronous with any other clocks in the design. When a clock is removed, it is removed from its related lists. To see if a clock is active or not, use the **report_clock** command with the **-attribute** option.

If a dynamic component of clock latency has been specified by **set_clock_latency** then the **-skew** option might be used to report what values have been set. In this case all components of clock latency will be.

Some information, such as the waveform of a generated clock or propagated skew, is displayed after a timing update (as a result of manually executing an update_timing function or experiencing an implicit timing update as a result of executing other commands). This behavior is intended to help you to define all clocks without incurring the cost of a complete timing update.

The report generated by using the **report_transitive_fanout** command with the **-clock_tree** option shows the fanout network of every clock source in the design.

To obtain a list of all clocks in the design, use the **all_clocks** command.

EXAMPLES

```
ptc_shell> create_clock -period 20.000000\
[get_ports {CLK}]

ptc_shell> report_clock
```

** Report : clock Design : counter Version: ... Date : Fri Jul 26 09:41:39 1996 ** Attributes: p - propagated_clock G - Generated clock Clock Period Waveform Attrs Sources ---------- ---------- ---------- ---------- CLK 20.00 {0 10} {CLK}

```
ptc_shell> set_clock_latency 2.4 [get_clocks CLK]
ptc_shell> set_clock_latency 0.17 -source -early [get_clocks CLK]
ptc_shell> set_clock_latency 0.19 -source -late  [get_clocks CLK]
ptc_shell> set_clock_uncertainty 1.3 [get_clocks CLK]
ptc_shell> set_clock_transition  1.7 [get_clocks CLK]
ptc_shell> report_clock -skew
```

** Report : clock_skew Design : counter Version: ... Date : Fri Jul 26 17:13:38 1996 ** Rise Fall Hold Setup Object Delay Delay Uncertainty Uncertainty ---------- ---------- ---------- ---------- ---------- CLK 2.40 2.40 1.3 1.3 ---------- Source Latency ---------- Object Min Rise Min Fall Max Rise Max Fall ---------- ---------- ---------- ---------- CLK 0.17 0.17 0.19 0.19 Rise Fall Object Transition Transition ---------- ---------- CLK 1.7 1.7

```
ptc_shell> set_clock_latency -late -source -dynamic 0.5 4.5 [get_clocks clk]
ptc_shell> set_clock_latency -early -source 2.5 -dynamic 0.5 [get_clocks clk]
ptc_shell> report_clock -skew
```

** Report : clock_skew Design : latency Version: ... Date : Tue Mar 20 08:14:26 2007 ** Min Condition Source Latency Max Condition Source Latency ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- Object Early_r Early_f Late_r Late_f Early_r Early_f Late_r Late_f Rel_clk ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- clk (static) 2.00 2.00 4.00 4.00 2.00 2.00 4.00 4.00 -- clk (dynamic) 0.50 0.50 0.50 0.50 0.50 0.50 0.50 0.50 -- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- clk (total) 2.50 2.50 4.50 4.50 2.50 2.50 4.50 4.50 --

```
ptc_shell> set_clock_groups -ex -group {clk1 clk3}
ptc_shell> set_clock_groups -asyn -name a1 -group clk2 -group clk2
ptc_shell> set_active_clocks {clk1 clk2}
ptc_shell> report_clock -groups
```

** Report : clock_groups Design : test Version: ... Date : Thu May 9 13:13:51 2002 ** Active clocks: clk1 clk2 Total exclusive groups: 1. NAME : clk1_others -group {clk1 clk3 } -group {clk2 clk4 } Total asynchronous groups: 1. NAME : a1 -group {clk2 } -group {clk4 }

SEE ALSO

```
all_clocks(2)
create_clock(2)
remove_clock_groups(2)
report_transitive_fanout(2)
set_active_clocks(2)
set_clock_groups(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
```

report_clock_crossing

Generates a report about paths launched from one clock and captured by another.

SYNTAX

```
string report_clock_crossing
   [-from from_list]
   [-to to_list]
   [-include include_list]
   [-format format]
   [-output file_name]
   [-nosplit]
   [-verbose]
   [-period]
   [-edge_relationships]
```

Data Types

```
   from_list     list
   to_list       list
   include_list  list
```

ARGUMENTS

-from *from_list*

Specifies a list of clocks to report clock crossing from. If no *-from* argument is given, all from clocks are reported.

-to *to_list*

Specifies a list of clocks to report clock crossing to. If no *-to* argument is given, all to clocks are reported.

-include *include_list*

Specifies a list of type of crossing to report. By default, all types are reported.

" valid - If some of the paths are **false** due to the **set_false_path** command, the interaction is still considered valid, but is marked with a '#'. " false - All of the paths are **false** due to the **set_false_path** command. The interaction is marked with an 'F'. " physically_exclusive - All of the paths are **false** due to a physically exclusive clock grouping. The interaction is marked with a 'P'. "

logically_exclusive - All of the paths are **false** due to a logically exclusive clock grouping. The interaction is marked with an 'L'. " asynchronous - All of the paths are **false** due to an asynchronous clock grouping. The interaction is marked with an 'A'.

Allowed values are as follows:

- *valid*:
  Reports clock interactions where there are valid paths.
  If some of the paths are **false**, that is noted with an attribute of '#'.

- *false*: Reports clock interactions where all of the paths between the clocks are declared as **false**. Only clock interactions that are connected with paths that are declared as **false** are reported. If there is a clock interaction declared as **false** but the design has no paths connecting those two clocks, that interaction is not reported.
- *logically_exclusive*: Reports clock interactions where all the paths between the clocks are declared as logically_exclusive with a **set_clock_groups** command. Only clock interactions that are connected with paths are reported.
- *physically_exclusive*: Reports clock interactions where all the paths between the clocks are declared as physically_exclusive with a **set_clock_groups** command. Only clock interactions that are connected with paths are reported.
- *asynchronous*: Reports clock interactions where all the paths between the clocks are declared as asynchronous with a **set_clock_groups** command. Only clock interactions that are connected with paths are reported.

-format *format*

Specifies the format of the report. The default for format must be either standard or csv. The default is standard. The default generates a text report. If csv is specified, the report is written out in a comma-separated value format. If csv is specified, the **-verbose** option is implied. This option requires the **-output** option.

-output *file_name*

Specifies the output file name for the output.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing tools to extract information from the report output.

-verbose

Adds more things to the report. One main addition is all pairs of clocks are added with "No paths" for the noninteracting clocks. In addition, the clock waveforms are also printed and minimum edge separation is printed separately when **-edge_relationships** is used.

-period

Adds the periods of each clock after each clock name and if there is a valid crossing the base period for the clocks is displayed. If there are no valid paths between the clocks, the base period is given as "--". If the base period is more than 101 times the smallest period, the base period is marked with "*". See the following example for more information.

-edge_relationships

Shows the most constraining edge relationship between the clocks that have valid crossing along with multicycle path information. This option also forces the **-period** option to be used so that the periods are displayed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Generates a report showing the clocks that launch datapaths that are capture by a different class. The from clock column gives the launching clock and the attribute column is blank for only valid paths. The following attributes are given to classify the interactions between the clocks in the row:

```
#     - some paths false
F     - all paths false
L     - logically exclusive clock group
P     - Physically Exclusive Clock Groups
A     - Asynchronous Clock Groups
A_MCP - all multicycle paths
S_MCP - some multicycle paths
A_MMD - all min-max delay paths
S_MMD - some min-max delay paths
MG    - from clock master of to clock
GM    - to clock master of from clock
N     - No valid path between the clock pair ( when -verbose option added )
V     - Valid path between the clock pair    ( when -verbose option added )
*     - Base Period Limit Exceeded ( when -period option is specified )
```

The Crossing Clocks column gives all of the clocks that capture paths from the "from clock" with the same attribute. The crossing clocks are separated by spaces and are split onto the next line unless the **-nosplit** option is given.

In addition to the default output, the *-format* and the *-output* options can be used to write the clock crossing report to a file in comma-separated value format.

EXAMPLES

The following example shows the summary report.

```
ptc_shell> report_clock_crossing -nosplit
****************************************
Report : report_clock_crossing
Design : test
Scenario: default
Version: ...
Date   : ...
****************************************

Attributes
    P     - physically exclusive clock groups
```

```
    L     - logically exclusive clock groups
    A     - asynchronous clock groups
    F     - all paths false
    #     - some paths false
    A_MCP - all multicycle paths
    S_MCP - some multicycle paths
    A_MMD - all min-max delay paths
    S_MMD - some min-max delay paths
    MG    - from clock master of to clock
    GM    - to clock master of from clock

From Clock           Attr     Crossing Clocks
------------------------------------------------------------------------------
CLK                  #,MG     GCLK
GCLK                          CLK2
GCLK                 A_MCP,GM CLK
```

The following example shows reporting the crossings from clock GCLK.

```
ptc_shell> report_clock_crossing -from GCLK -nosplit
****************************************
Report : report_clock_crossing
Design : test
Scenario: default
Version: ...
Date   : ...
****************************************

Attributes
    P     - physically exclusive clock groups
    L     - logically exclusive clock groups
    A     - asynchronous clock groups
    F     - all paths false
    #     - some paths false
    A_MCP - all multicycle paths
    S_MCP - some multicycle paths
    A_MMD - all min-max delay paths
    S_MMD - some min-max delay paths
    MG    - from clock master of to clock
    GM    - to clock master of from clock

From Clock           Attr      Crossing Clocks
------------------------------------------------------------------------------
GCLK                           CLK2
GCLK                 A_MCP,GM  CLK
```

The following example shows using the **-period** option. Note that the base period of "--" is given if there are no valid paths. The base period is marked with "*" if it exceeds 101 times the smallest period. Also, note that this option forces only two clocks to be reported per line.

```
ptc_shell> report_clock_crossing -period -nosplit
****************************************
Report : report_clock_crossing
Design : test1
Scenario: default
Version: ...
Date   : ...
****************************************

Attributes
    P     - physically exclusive clock groups
    L     - logically exclusive clock groups
    A     - asynchronous clock groups
    F     - all paths false
```

```
    #     - some paths false
    A_MCP - all multicycle paths
    S_MCP - some multicycle paths
    A_MMD - all min-max delay paths
    S_MMD - some min-max delay paths
    MG    - from clock master of to clock
    GM    - to clock master of from clock
    *     - Base Period Limit Exceeded

Base
Period  From Clock           Attr   Crossing Clocks
------------------------------------------------------------------------------
10.64   CLK(5.32)            MG     GCLK(10.64)
1143.32* CLK3(11.32)         MG     GCLK(10.64)
--      GCLK(10.64)          F      CLK2(15.00)
10.64   GCLK(10.64)          GM     CLK(5.32)
1143.32* GCLK(10.64)         GM     CLK3(11.32)
```

The following example shows using the **-edge_relationships** option. Note that for clocks that have valid clock crossing it shows the most constraining edge relationship.

```
ptc_shell> report_clock_crossing -edge_relationships -nosplit
****************************************
Report : report_clock_crossing
Design : test
Scenario: default
Version: ...
Date   : ...
****************************************

Attributes
    P     - physically exclusive clock groups
    L     - logically exclusive clock groups
    A     - asynchronous clock groups
    F     - all paths false
    #     - some paths false
    A_MCP - all multicycle paths
    S_MCP - some multicycle paths
    A_MMD - all min-max delay paths
    S_MMD - some min-max delay paths
    MG    - from clock master of to clock
    GM    - to clock master of from clock
    *     - Base Period Limit Exceeded

Base
Period  From Clock   Attr      Crossing Clocks   Setup edge Hold edge
------------------------------------------------------------------------------
20.00   CLK(10.00)   #,MG      GCLK(20.00)       R(--)->R(--)[MCP 1]
                                                 R(0.00)->R(0.00)[MCP 0]
--      GCLK(20.00)  F         CLK2(15.00)
20.00   GCLK(20.00)  A_MCP,GM  CLK(10.00)        R(0.00)->R(30.00)[MCP 3]
                                                 R(0.00)->R(20.00)[MCP 0]
20.00   GCLK(20.00)  A_MCP,GM  CLK(10.00)        F(10.00)->R(40.00)[MCP 3]
                                                 F(10.00)->R(30.00)[MCP 0]
```

SEE ALSO

```
analyze_paths(2)
get_clock_crossing_points(2)
```

report_clock_gating_check

Displays clock gating check information about specified pins.

SYNTAX

```
int report_clock_gating_check
   [-significant_digits digits]
   [-nosplit]
   [object_list]

list object_list
```

ARGUMENTS

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. **-nosplit** prevents line splitting and facilitates writing software to extract information from the report output.

object_list

Specifies a list of cells, pins, clocks or design objects for report.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **report_clock_gating_check** command generates two tables. The first table displays information

about all the clock gating checks, including cell, enable pin, clock pin, level, clock (check w.r.t. a specific clock, or all clocks if none is given), attribute (inferred, or icg), and what user settings have impacted its S/H/R/F values and check level. The attribute indicates where the clock gating check was originally defined. It can be either inferred, or defined by library cell.

If there are multiple settings impacting a clock gating check, it is an error condition. The conflicts should be resolved by the user.

The second table displays information about user settings that impact the clock gating checks displayed in the first table, including netlist object of clock it is set on, S/H/R/F check values, level, and file/line where the relevant **set_clock_gating_check** and **remove_clock_gating_check** commands are defined. If there is no effective user settings impacting clock gating checks, the second table will not display.

EXAMPLES

The following example displays how to report all clock gating check informations.

```
ptc_shell> report_clock_gating_check
```

report_clock_gating_check ** Report : clock_gating_check Design : cgbox Scenario: default Version: ... Date : Fri Nov 14 11:02:20 2008 ** Updating Clock Gating Locations Inferring 6 clock-gating checks. Clock gating checks: Cell EnablePin ClockPin High/Low Clock Attr User Settings --- U4 E CK Low L 1 U1 A B High I 2 U2 B A High I 2 U3 A B High I 3 U5 A B High I 3 U9 S0 B Low I 4 U9 S0 A Low clk1 I 5 Attr: I:auto inferred, L:library cell defined User clock gating settings: Rise Fall ID Object Setup Hold Setup Hold High/Low File/Line --- 1 U4/CK -- -- -- -- Low clock_gating2.tcl, line 79 2 U1/A 4.00 3.00 3.00 3.00 -- clock_gating2.tcl, line 87; clock_gating2.tcl, line 96 3 U8/Y 2.00 2.00 2.00 2.00 -- clock_gating2.tcl, line 71 4 U9/S0 -- -- -- -- Low clock_gating2.tcl, line 51 5 clk1 1.00 1.00 1.00 1.00 -- clock_gating2.tcl, line 61 1

SEE ALSO

set_clock_gating_check(2)
remove_clock_gating_check(2)

report_clock_map

Synonym for the "report_clock -map" command.

SEE ALSO

report_clock(2)

report_constraint_analysis

Reports constraint statistics, rule violations, user messages, and information about defined rules.

SYNTAX

```
string report_constraint_analysis
   [-include include_list]
   [-style style]
   [-format format]
   [-output file_name]
   [-rules rule_list]
   [-rule_types type_list]
```

Data Types

```
   include_list      list
   type_list         list
   rule_list         list
```

ARGUMENTS

-include *include_list*

Specifies the content to include in the output. The include list can contain any of the following: statistics, violations, user_messages, and rule_info. You can use this option to report a combination of constraint processing statistics, violations, user messages, or rule information without needing to report all four.

-style *style*

Specifies the style of report to print. Possible values for style are: **summary**, **full**. The default is **full**. A description and example of the different styles is given below.

-format *format*

Specifies the format of the report. The values for format must either be standard or csv. The default is standard, which generates a text report. If csv is specified, the violations are written out in a comma separated value format. This option requires the **-output** option. If this option is specified, the **statistics**, and **rule_info** values cannot be specified with the **-include** option.

-output *file_name*

Specifies the output file name for the output.

-rules ***rule_list***

Specifies that the output of the report should only contain the rules in the rule list and not all the rules available. You can combine this option with the **-include** option to further filter what categories need to be included. The statistics and user_messages categories do not correspond to a specific set of rules and are included in totality.

-rule_types ***type_list***

Specifies that the output of the report should only contain the corresponding rules of the given rule types and not all the rules available. The allowed values are analyze_design, block2top and sdc2sdc. The analyze_design value includes all the rules checked in the analyze_design command. The block2top value includes all the block to top rules that are checked in the compare_block_to_top command. The sdc2sdc value includes all the constraint comparison rules checked in the compare_constraints command. You can combine this option with the **-include** option to further filter what categories are included. The statistics and user_messages categories do not correspond to a specific set of rules, and they are included in totality. This option cannot be used with the **-rules** option because **-rules** already specifies a set of rules.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reports constraint statistics, rule violations, user messages, and information about defined rules. By default, the command produces a report of the constraints read into the session and whether they were accepted by the tool, all the violations that occurred in the analysis, a report of all user messages that occurred, and information about all defined rules.

Use the *-include* option to select a subset of the four reports shown by default.

You can also limit the rules reported by using the *-rules* and *-rule_types* options. The *-rules* option can be used to specify a set of rules to be reported. The *-rule_types* option can be used to specify the types of rules to be reported.

By default, for reporting violations, all violation instances are fully instantiated. The *-style* option can be used to write out the violation counts and not all the violation instances.

In addition to the default output, the *-format* and the *-output* options can be used to write the violations report to a file in comma separated value format.

EXAMPLES

This example shows the default report.

```
ptc_shell> report_constraint_analysis

****************************************
Report : report_constraint_analysis
Version: ...
Date   : Wed Aug 13 16:41:02 2008
****************************************

Constraint Processing Statistics

SDC Command                        Accepted     Rejected     Total
------------------------------------------------------------------------------
create_clock                       3            0            3
set_input_delay                    6            0            6
  ....

Scenario Violations  Count Waived Description
------------------------------------------------------------------------------
<Netlist Violations>    1    0    Scenario independent violations
  error                 1    0
    UNT_0002            1    0    Library 'library' has incomplete units defined.
      1 of 1                0    Library 'lsi_10k' has incomplete units defined.
------------------------------------------------------------------------------
Total Error Messages    1    0

User Messages

MessageID            Count   Description
------------------------------------------------------------------------------
Error                1
  CMD-005            1       unknown command '%s'
    1 of 1           1       unknown command 'foo'
Information          3
  CMP-006            3       Reading %s configuration file: %s
    1 of 3           1       Reading PrimeTime configuration file:
                             /remote/srm283/clientstore/main_gca_build1/snps/ \
                             synopsys/auxx/gca/tcl/primetime.pcx
   ....


Rule Information

Rule          Severity  Status    Message
------------------------------------------------------------------------------
NTL_0001      warning   enabled   Output port 'port' is unbuffered.
NTL_0002      error     enabled   Net 'net' has both strong and three-state drivers.
NTL_0003      warning   enabled   Net 'net' has potentially invalid multiple strong drivers.
NTL_0004      info      enabled   Net 'net' is a top-level feedthrough.
NTL_0005      warning   enabled   Unresolved reference to 'reference_name'.
1
```

This example shows the summary report for only violations.

```
ptc_shell> report_constraint_analysis -style summary -include violations
****************************************
Report : report_constraint_analysis
        -include {violations }
        -style {summary}
****************************************

Scenario Violations  Count Waived Description
------------------------------------------------------------------------------
<Netlist Violations>    1    0    Scenario independent violations
```

```
  error                1    0
```

This example writes the violation report to a file in CSV format.

```
ptc_shell> report_constraint_analysis -format csv -output ./1.out
****************************************
Report : report_constraint_analysis
        -include {violations }
        -output ./1.out
        -format {csv}
        -style {summary}
****************************************
```

This example writes the violation report for one specific rule UNT_0002.

```
ptc_shell> report_constraint_analysis -rules {UNT_0002} -include violations

****************************************
Report : report_constraint_analysis
Version: ...
Date   : Wed Aug 13 16:41:02 2008
****************************************

Scenario Violations  Count Waived Description
--------------------------------------------------------------------------------
<Netlist Violations>    1    0    Scenario independent violations
  error                 1    0
    UNT_0002            1    0    Library 'library' has incomplete units defined.
      1 of 1                0    Library 'lsi_10k' has incomplete units defined.
--------------------------------------------------------------------------------
Total Error Messages    1    0
```

SEE ALSO

analyze_design(2)
report_rule(2)
compare_constraints(2)
compare_block_to_top(2)

report_data_check

Reports user specified data-to-data checks in the design.

SYNTAX

```
string report_data_check
   [-nosplit]
```

Data Types

```
   -nosplit   boolean
```

ARGUMENTS

-nosplit

Specifies not to split lines if a column overflows.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reports the user-defined data-to-data constraints applied to the design.

EXAMPLES

The following example specifies and reports a data-to-data check.

```
ptc_shell> set_data_check -rise_from [get_pin ffa/D] -rise_to [get_pin ffb/D] -setup 1.0 -hold 1.0
```

```
ptc_shell> report_data_check -nosplit
****************************************
Report : data_check
Design : simple_2ff
Version: G-2012.06
Date   : Mon Apr  9 13:50:15 2012
****************************************

From                    To                      Clock       Hold     Setup
                                                            Value    Value
--------------------------------------------------------------------------------
ffa/D(r)                ffb/D(r)                -            1.000   1.000
1
```

SEE ALSO

set_data_check(2)
remove_data_check(2)

report_design

Displays attributes on the **current_design**.

SYNTAX

```
int report_design [-nosplit]
```

ARGUMENTS

-nosplit

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. This option prevents line-splitting and facilitates writing software to extract information from the report output.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Lists information about the attributes on the **current_design**.

EXAMPLES

The following is an example of a design report.

```
ptc_shell> report_design
```

********** Report : design Design : counter Version: ... Date : Fri Aug 2 17:12:27 1996 ********** Design Attribute Value ---------- ---------- Operating Conditions: operating_condition_max_name WCCOM process_max 1.50 temperature_max 70.00 voltage_max 4.75 tree_type_max worst_case_tree Wire Load: (use report_wire_load for more information) wire_load_mode top wire_load_model_max -- wire_load_selection_group_max -- wire_load_min_block_size 0 Design Rules: max_capacitance 0.5 min_capacitance -- max_fanout 5.6 min_fanout -- max_transition 0.8 min_transition -- max_area -- Timing Ranges: fastest_factor -- slowest_factor --

SEE ALSO

report_clock(2)
report_port(2)

report_design_mismatch

Reports all design mismatches found during link.

SYNTAX

```
int report_design_mismatch
   [-message_display_limit limit]
   [-nosplit]
```

ARGUMENTS

-message_display_limit

Specifies the limit on the number of user messages per each message type. If this option is not provided, a default of 20 is used.

-nosplit

This option prohibits breaking of lines in the reports when columns overflow. 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. The -nosplit option prevents line-splitting and facilitates writing tools to extract information from the report output.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

When **link_allow_design_mismatch** is true, the linker in constraint consistency succeeds even when mismatches are found during link. This allows you to gather useful information even when some parts of the design are in the early development stages. The **report_design_mismatch** command provides a report of mismatches for the current design.

Common causes of mismatches include:

```
1). A pin of an instance does not exist in the reference.
2). Bus widths of the instance and the reference are different.
```

These two causes are in increasing order of priority.

If a pin has any mismatched issues, the "**is_design_mismatch**" attribute is set on the pin along with its connected net and cell. If the mismatch occurs on a bus, all other bits of the bus are marked with this attribute.

Note that if a cell or net has more than one mismatched issue, only the one with the highest priority is reported.

EXAMPLES

The following example shows a report of design mismatch information for a design which instantiate another module with connections of incorrect widths. It also contains connections to nonexisting pins of a design.

```
ptc_shell> report_design_mismatch -class pin
report_design_mismatch
****************************************
Report : design_mismatch
Design : top
Version: F-2011.12-Beta2
Date   : Wed Oct 12 11:11:36 2011
****************************************

User Messages

Message   Design    Object    Type    Count       Mismatch Description
-------------------------------------------------------------------------------------------
  DMM-038 design    object    type     4          Pin "%s" of cell "%s" of reference "%s" in
                                                  design "%s" shows inconsistency between master
                                                  and instantiation because %s. Ignoring this pin.
          top       U8/GND    pin      1 of 4     Pin "GND" of cell "U8" of reference "top" in
                                                  design "BUFFD1" shows inconsistency between
                                                  master and instantiation because it does not
                                                  exist in the reference block. Ignoring this pin.
          top       U8/PWR    pin      2 of 4     Pin "PWR" of cell "U8" of reference "top" in
                                                  design "BUFFD1" shows inconsistency between
                                                  master and instantiation because it does not
                                                  exist in the reference block. Ignoring this pin.
          top       U8/GND    pin      3 of 4     Pin "GND" of cell "U8" of reference "top" in
                                                  design "BUFFD1" shows inconsistency between
                                                  master and instantiation because it does not
                                                  exist in the reference block. Ignoring this pin.
Information: Reaching the message display limit of report_design_mismatch. (UIC-063)

Message   Design    Object    Type    Count       Mismatch Description
-------------------------------------------------------------------------------------------
  DMM-904 design    object    type     4          The width [%d] of '%s' bus on '%s' cell in '%s'
                                                  design is mismatched with the width [%d] of '%s'
                                                  bus on the '%s' reference.
          top       u_block   cell     1 of 4     The width [2] of 'data_in' bus on 'u_block' cell
                                                  in 'top' design is mismatched with the width [12]
                                                  of 'data_in' bus on the 'block' reference.
          top       u_block   cell     2 of 4     The width [2] of 'data_out' bus on 'u_block' cell
                                                  in 'top' design is mismatched with the width [6]
                                                  of 'data_out' bus on the 'block' reference.
```

```
              top       u_block   cell     3 of 4     The width [2] of 'data_in' bus on 'u_block' cell
                                                      in 'top' design is mismatched with the width [12]
                                                      of 'data_in' bus on the 'block' reference.
    Information: Reaching the message display limit of report_design_mismatch. (UIC-063)
```

SEE ALSO

link_allow_design_mismatch(3)

report_disable_timing

Reports disabled timing arcs in the current design.

SYNTAX

```
string report_disable_timing
   [-nosplit]
   [-all_arcs]
   [cells_or_ports]
```

collection *cells_or_ports*

ARGUMENTS

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, 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.

-all_arcs

Print the enabled as well as the disabled timing arcs. This option may only be used if cells objects are given. Enabled timing arcs will be labeled with the string "enabled" in the flags column.

cells_or_ports

Limits disabled arc reporting to the specified list of cells or ports. Provide the list or collection of cells or ports as an argument to the command.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Reports disabled timing arcs in the current design. Timing arcs can be disabled in several ways. You can

disable a timing arc by using the **set_disable_timing** command. The following symbols are used to explain why a timing arc is disabled:

c : The propagation of case-analysis values

C : The presence of conditional arcs (arcs that have a when statement defined in the library)

d : The presence of default arcs (when the **timing_disable_cond_default_arcs** variable is set to true)

f : Disabling of false net-arcs

l : Loop breaking

L : db inherited loop breaking

p : The propagation of constant values

u : Arcs disabled by using the **set_disable_timing** command.

EXAMPLES

The following example shows that the timing arc of a cell named *U1/U2* from pin *A* to pin *Z* is disabled.

```
ptc_shell> set_disable_timing {n2_i} -from A -to Z
ptc_shell> report_disable_timing
```

```
****************************************
Report : disable_timing
Design : middle
****************************************

Flags :     c  case-analysis
            C  Conditional arc
            d  default conditional arc
            f  false net-arc
            l  loop breaking
            L  db inherited loop breaking
            p  propagated constant
            u  user-defined

Cell or Port         From    To      Sense           Flag  Reason
------------------------------------------------------------------------
n2_i                 A       Z       negative_unate  u

1
```

The following example specifies that the timing arc of cell *ff4* from pin *CP* to pin *TE* and *TI* are disabled as a result of a case-analysis constant of *0* propagated to pin *TE* of cell *ff4*. Note that the actual case value is set on another pin *A* and the propagation path to *ff4/TE* is inverting.

```
ptc_shell> set_case_analysis 0 {p_in in0}
ptc_shell> report_disable_timing

****************************************
Report : disable_timing
```

```
Design : middle
****************************************

Flags :     c  case-analysis
            C  Conditional arc
            d  default conditional arc
            f  false net-arc
            l  loop breaking
            L  db inherited loop breaking
            p  propagated constant
            u  user-defined

Cell or Port          From    To      Sense           Flag  Reason
-------------------------------------------------------------------------
o_reg4                CP      D       hold_clk_rise   c     D = 0
o_reg4                CP      D       setup_clk_rise  c     D = 0
o_reg4                CP      CD      recovery_rise_clk_rise
                                                      c     D = 0
p_out                                                 c     p_out = 1

1
```

To view disabled arcs in specific cells, provide a list or collection of the required cells as an argument to the command. In the above example, to view disabled arcs for *o_reg4* instance, do the following:

```
ptc_shell> report_disable_timing  [get_cells { o_reg4 }]

****************************************
Report : disable_timing
Design : middle
****************************************

Flags :     c  case-analysis
            C  Conditional arc
            d  default conditional arc
            f  false net-arc
            l  loop breaking
            L  db inherited loop breaking
            p  propagated constant
            u  user-defined

Cell or Port          From    To      Sense           Flag  Reason
-------------------------------------------------------------------
o_reg4                CP      D       hold_clk_rise   c     D = 0
o_reg4                CP      D       setup_clk_rise  c     D = 0
o_reg4                CP      CD      recovery_rise_clk_rise
                                                      c     D = 0

1
```

SEE ALSO

remove_disable_timing(2)
set_disable_timing(2)

report_exceptions

Generates a report of timing exceptions.

SYNTAX

```
Boolean report_exceptions
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-ignored]
   [-remove_invalid_start_end_points]
   [-dominant]
   [-verbose]
   [-nosplit]

list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
```

ARGUMENTS

Note: Options marked with asterisks (*) in the SYNTAX can be used more than once in the same command.

-from *from_list*

Specifies a list of clocks, ports, cells, and pins in the current design. The report is to include only exceptions that have the given objects in the *from_list*. Using this option limits the report to information that was set using a path-based command (for example, **set_multicycle_path**) with the **-from** option. The **-from**, **-rise_from**, and **-fall_from** options are mutually exclusive.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified.If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins or ports. The report is to include only paths that go through the pins or ports on *through_list*. Using this option limits the report to information that was set using a path-based command (for example, **set_multicycle_path**) with the **-through** option. You can use this option more than once in the same command.

-rise_through ***rise_through_list***

Specifies a list of pins or ports (the same as the **-through** option) except that the paths must have a rising transition at the through points. You can use this option more than once in the same command.

-fall_through ***fall_through_list***

Specifies a list of pins or ports (the same as the **-through** option) except that the paths must have a falling transition at the through points. You can use this option more than once in the same command.

-to ***to_list***

Specifies a list of clocks, ports, cells, and pins in the current design. The report is to include only paths that end at the objects in the *to_list* objects. Using this option limits the report to information that was set using a path-based command (for example, **set_multicycle_path**) with the **-to** option. The **-to**, **-rise_to**, and **-fall_to** options are mutually exclusive.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-ignored

Indicates that the report is to lists path timing exceptions that are set on the current design, but are completely ignored. For example, a false path might be specified from port A to port Z1, but if there is no timing path between those points, the path is ignored. Use **-from** or **-to** to limit the report to certain paths.

-dominant

Indicates that the report is to lists path timing exceptions that are set on the current design and are dominant for at least one path. This option may be used with the -ignored option to see all exceptions. If neither the -ignored or -dominant option are given all exceptions will be printed whether they effect and paths or not.

-remove_invalid_start_end_points

Use this option to prune the invalid start and end points from exceptions. If all the start or end points of an exception are invalid, then the entire exception is pruned from the report. You can combine this option along with -dominant option to write out a compacted list of exceptions affecting the design. This option is mutually exclusive with -ignored option.

-verbose

Use with the **-ignored** option. Reports each dominant exception that overrides each ignored exception.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Produces a report showing information about timing exceptions for the current design.

Constraint consistency accepts a timing exception as dominant if it alters the constraints of a path. If path constraints are different in the absence of a timing exception, the exception alters or dominants the path constraint.

The report_exceptions command attempts to identify reasons an exception is ignored, and classifies these into one of the following: invalid startpoints, invalid endpoints, non-existent paths, or overridden paths. This is reported as separate sections of the report. When -ignored is used, the sections will show why an exception was ignored.

If a path is unconstrained (that is, if it has a required time of infinity), timing exceptions do not change the path constraints. Applying a timing exception to an unconstrained path does not change the path constraint. A path is unconstrained if it has an infinite required time.

To remove timing exceptions from specified paths, use **reset_path**. **reset_design** removes all attributes from the design, including timing exceptions.

To see only those exceptions that are dominant use the -dominant option. If neither the -ignored or -dominant option are given then all exceptions matching the -from/-through/-to options will be given and not categorized into ignored and dominant sections.

EXAMPLES

The following example lists all timing exceptions set on the design.

```
ptc_shell> report_exceptions
```

****
Report : exceptions
Design : counter
Scenario: default
Version: ...
Date : Fri Nov 14 15:31:16 2008
****
e8.tcl, line 9; e8.tcl, line 10
set_multicycle_path 2 -hold -from [get_clocks {clk1}] -to [get_clocks {clk2}]
set_multicycle_path 5 -setup -from [get_clocks {clk1}] -to [get_clocks {clk2}]
e8.tcl, line 11
set_false_path -from [get_clocks {clk1}] -to [get_clocks {clk2}]
e8.tcl, line 13
set_multicycle_path 2 -through [get_pins {t/Z}]
e8.tcl, line 14
set_false_path -through [get_pins {t/A}]
e8.tcl, line 16
set_multicycle_path 2 -through [get_pins {u/Z}] -through [get_pins {u/B}]
e8.tcl, line 21
set_false_path -through [get_pins {t/Z}]

The following example lists all ignored timing exceptions set on the design.

```
ptc_shell> report_exceptions -ignored
```

****
Report : exceptions
Design : counter
Scenario: default
Version: ...
Date : Fri Nov 14 15:31:16 2008
****
##########
Redundant exceptions (totally overridden by other exceptions):
e8.tcl, line 9; e8.tcl, line 10
set_multicycle_path 2 -hold -from [get_clocks {clk1}] -to [get_clocks {clk2}]
set_multicycle_path 5 -setup -from [get_clocks {clk1}] -to [get_clocks {clk2}]
##########
Exceptions that do not cover any constrained paths:
e8.tcl, line 16
set_multicycle_path 2 -through [get_pins {u/Z}] -through [get_pins {u/B}]
##########
Exceptions with no valid -from objects:
e8.tcl, line 24
set_false_path -from [get_pins {u/B}]

The following example lists all dominant timing exceptions set on the design.

```
ptc_shell> report_exceptions -dominant
```

****
Report : exceptions
Design : counter
Scenario: default
Version: ...
Date : Fri Nov 14 15:46:16 2008
****
##########
Exceptions that are dominant for at least one path:
e8.tcl, line 11
set_false_path -from [get_clocks {clk1}] -to [get_clocks {clk2}]
e8.tcl, line 13
set_multicycle_path 2 -through [get_pins {t/Z}]
e8.tcl, line 14
set_false_path -through [get_pins {t/A}]

The following example shows how the -verbose option shows the overriding exceptions along with the ignored exceptions

```
ptc_shell> report_exceptions -ignored -verbose
```

****
Report : exceptions
Design : test
Scenario: default
****
##########
Redundant exceptions (totally overridden by other exceptions):
The following exception in test.tcl at line 62 is dominated by other exceptions:
set_multicycle_path 2 -setup -from [get_clocks {CLK1_1}]
The following exceptions dominate the above exception :
test.tcl, line 62
set_multicycle_path 3 -setup -from [get_pins {H1/H1/F1/CLK}]
test.tcl, line 64
set_multicycle_path 2 -setup -from [get_pins {H2/H1/F1/CLK}]

SEE ALSO

current_design(2)
set_false_path(2)
set_max_delay(2)
set_max_time_borrow(2)
set_min_delay(2)
set_multicycle_path(2)

report_lib

Reports library information.

SYNTAX

```
string report_lib
   [-nosplit]
   [-timing_arcs]
   library
   [lib_cell_list]
```

Data Types

```
   library              list
   lib_cell_list             list
```

ARGUMENTS

-nosplit

Prevents splitting lines if a column overflows. Some of the 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. This option prevents line-splitting and facilitates writing software to extract information from the report output.

-timing_arcs

Displays timing arc information. Indicates to list all cell timing arcs.

library

A pattern matching a single library or a collection containing one library. The library must have been read by using the **read_db** command.

lib_cell_list

Displays a list of library cells about which information is reported. The default is to report information about all cells in the technology library. Each element in this list is either a collection of library cells or a pattern that matches library cells in the *library* option.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

A library report displays library information including a list of operating conditions, wire load models, and available library cells. The *-timing_arcs* option lists detailed timing arc information for each library cell.

To generate a report, the specified library must be loaded into ptc_shell.

EXAMPLES

This example shows the default library report.

```
ptc_shell> read_db tech_lib.db

ptc_shell> report_lib tech_lib

****************************************
Report : library
Library: tech_lib
****************************************

Time Unit                    : 1 ns
Capacitance Unit             : 1 pF

Operating Conditions:

   Name                 Process     Temp        Voltage     Tree Type
   --------------------------------------------------------------------------
   BCCOM                  0.60        0.00        5.25     best_case
   TYPICAL                1.00       25.00        5.00     balanced_case
   WCCOM                  1.50       70.00        4.75     worst_case


Library Cells:

   Attributes:
    b - black box (function unknown)

   Lib Cell  Attributes
   ------------------------------
   AN2
   BUFA
   BUFB
   FF
   IV
   LTCH
   MUX21
   OR2
```

In this example, timing arc information is shown for a list of library cells.

```
ptc_shell> report_lib -timing_arcs tech_lib {AN2 OR2}

****************************************
Report : library
         -timing_arcs
Library: tech_lib
****************************************

Library Cells:

   Attributes:
    b - black box (function unknown)

                             Arc                    Arc Pins
   Lib Cell  Attributes    #  Sense/Type      From         To          When
   ------------------------------------------------------------------------------
   AN2                     0  unate           A            Z
                           1  unate           B            Z

   OR2                     0  unate           A            Z
                           1  unate           B            Z
```

SEE ALSO

read_db(2)

report_mode

Displays a report of modes for specified cells.

SYNTAX

```
string report_mode
   [-nosplit]
   [instance_list]
```

Data Types

```
   instance_list      list
```

ARGUMENTS

-nosplit

Most of the mode 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. This option prevents line-splitting and facilitates writing tools to extract information from the report output.

instance_list

Specifies a list of instances whose modes are to be reported. By default, the report includes all cells that have modes. This option is only valid if the *-type* argument has a cell value.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command reports cell modes. The report specifies whether the cell mode is enabled or disabled, and it also specifies the reason why the mode is enabled or disabled. There are three possible reasons why a mode can be enabled or disabled. They are

- cell - Indicates that the cell mode has been set directly using the **set_mode** command.
- cond - Indicates that the cell mode has been set due to the evaluation of the mode condition during constant propagation.
- default - Indicates that the cell mode has not been set by any of the preceding reasons.

EXAMPLES

The following example reports cell mode information for the design top_design. Two cells have modes, Uram1 and Uram2 are instances of the library cell RAM2_core. rw is a cell mode group defined on the library cell RAM2_core. This cell mode group has two cell modes read and write. No modes are currently active, so all modes are enabled and the reason is given as default.

```
ptc_shell> report_mode

****************************************
Report : mode
Design : top_design
****************************************

Cell                              Mode(Group)       Status      Reason
--------------------------------------------------------------------------------
Uram1/core(RAM2_core)             read(rw)          ENABLED     default
                                  write(rw)         ENABLED     default
--------------------------------------------------------------------------------
Uram2/core(RAM2_core)             read(rw)          ENABLED     default
                                  write(rw)         ENABLED     default
--------------------------------------------------------------------------------
```

The following example shows a report of modes specified for the two RAMs that have modes in the design. Ram Uram1 is set in mode read, and Ram Uram2 is set in mode write. Therefore, all timing arcs of RAM Uram1 associated with mode read are enabled, and all timing arcs associated with mode write are disabled.

```
ptc_shell> set_mode read Uram1/core
ptc_shell> set_mode write Uram2/core
ptc_shell> report_mode

****************************************
Report : mode
Design : top_design
****************************************

Cell                              Mode(Group)     Status         Reason
--------------------------------------------------------------------------------
Uram1/core(RAM2_core)             read(rw)        ENABLED        cell
                                  write(rw)       disabled       cell
--------------------------------------------------------------------------------
Uram2/core(RAM2_core)             read(rw)        disabled       cell
                                  write(rw)       ENABLED        cell
--------------------------------------------------------------------------------
```

SEE ALSO

```
reset_mode(2)
set_mode(2)
```

report_net

Generates a report of net information.

SYNTAX

```
string report_net
   [-connections [-verbose]]
   [-significant_digits digits]
   [-segments]
   [-nosplit]
   [net_names]

list net_names
```

ARGUMENTS

-connections

Indicates that the report is to show net connection information.

-verbose

Must be used with the **-connections** option. Indicates that the report is to show verbose connection information.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. Fixed-precision floating-point arithmetics is used for delay calculation; therefore, the actual precision might depend on the platform. For example, on SVR4 UNIX see FLT_DIG in limits.h. The upper bound on a meaningful value of the argument to **-significant_digits** is then FLT_DIG - ceil(log10(fabs(*any_reported_value*))).

-segments

Indicates that the report is to show all global segments for each net requested. Global net segments are all those physically connected across all hierarchical boundaries.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field

exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

net_names

Specifies a list of nets to be reported. The default is to report all nets in the current instance or current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The command displays information about the nets in the design of the current instance, or in the current design. If **current_instance** is set, the report is generated for the design of that instance; otherwise, the report is generated for the current design. Attributes, such as annotated resistance and annotated capacitance, are displayed.

If **-connections** is used, constraint consistency lists the leaf cell pins connected to each net.

EXAMPLES

The following example reports all nets in the design.

```
ptc_shell> report_net

****************************************
Report : net
Design : top
Version: ...
Date   : Wed Aug  7 09:28:05 1996
****************************************

Attributes:
    c - annotated capacitance
    r - annotated resistance

Net                  Fanout     Fanin       Cap   Resistance    Pins   Attributes
--------------------------------------------------------------------------------
a                         2         1      2.00         0.00       3
b                         1         1      1.00         0.00       2
c                         1         1      1.00         0.00       2
d                         1         1      1.00         0.00       2
e                         1         1      1.00         0.00       2
en                       20         1     25.00         0.00      21
f                         1         1      1.00         0.00       2
g                         1         1      1.00         0.00       2
h                         1         1      1.00         0.00       2
i1                        1         1      1.00         0.00       2
i2                        1         1      1.00         0.00       2
```

```
i3                        1         1      1.00         0.00       2
i4                        1         1      1.00         0.00       2
j                         1         1      1.00         0.00       2
k                         1         1      1.00         0.00       2
l                         1         1      1.00         0.00       2
m                         2         1      2.00         0.00       3
n                         1         1      1.00         0.00       2
o                         1         1      1.00         0.00       2
o1                        1         1      0.00         0.00       2
o2                        1         1      0.00         0.00       2
p                         1         1      1.00         0.00       2
q                         2         1      2.00         0.00       3
r                         1         1      1.00         0.00       2
s                         1         1      1.00         0.00       2
t                         1         1      1.00         0.00       2
u                         1         1      1.00         0.00       2
w                         1         1      1.00         0.00       2
y                         1         1      1.00         0.00       2
z                         1         1      2.00         0.00       2
-------------------------------------------------------------------------------
Total 30 nets            52        30     56.00         0.00      82
Maximum                  20         1     25.00         0.00      21
Average                1.73      1.00      1.87         0.00    2.73
```

The following example displays a verbose connection report for net z.

```
ptc_shell> report_net -verbose -connections z

****************************************
Report : net
         -connections
         -verbose
Design : top
Version: ...
Date   : Wed Aug  7 09:30:38 1996
****************************************


Connections for net 'z':
    pin capacitance:    2
    wire capacitance:   0
    total capacitance:  2
    wire resistance:    0
    number of drivers:  1
    number of loads:    1
    number of pins:     2

    Driver Pins              Type                 Pin Cap
    ------------             ----------------     --------
    z/Z                      Output Pin (NOR2)     0

    Load Pins                Type                 Pin Cap
    ------------             ----------------     --------
    o2/B                     Input Pin (BUF)       2
```

SEE ALSO

```
current_design(2)
current_instance(2)
report_cell(2)
report_design(2)
```

report_path_group

Reports user-defined path_group information.

SYNTAX

```
string report_path_group
   [-nosplit]
   [path_group_list]
```

Data Types

path_group_list	list

ARGUMENTS

-nosplit

Does not split lines if a column overflows.

path_group_list

Displays the path group information for the *path_group_list* that is specified. Constraint consistency supports only path group names and not collection.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Produces a report showing information about user-defined path groups in the **current_design** command.

EXAMPLES

The following command displays all of the path groups in the current design.

```
ptc_shell> report_path_group

****************************************
Report : path_group
Design : test
Version: A-2007.12-DEV
Date   : Sun Jul 29 12:03:16 2007
****************************************

Path_Group     Weight   From      Through         To
------------------------------------------------------------
**default**      1.00   -         -               -
tata             3.00   *         *               C1
```

SEE ALSO

```
create_clock(2)
current_design(2)
group_path(2)
reset_design(2)
```

report_port

Displays port information within the design.

SYNTAX

```
string report_port [-verbose]
   [-drive]
   [-input_delay]
   [-output_delay]
   [-nosplit]
   [port_names]

list port_names
```

ARGUMENTS

-verbose

Indicates that the port report includes all port information. By default, only a summary section is displayed that lists all ports and their direction.

-drive

Reports only drive resistance, input transition time, and driving cell information for only input and inout ports.

-input_delay

Reports only the port input delay information you set.

-output_delay

Reports only the port output delay information you set.

-nosplit

Prevents line splitting if column overflows. Most 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. This option prevents line-splitting and facilitates writing software to extract information from the report output.

port_names

Displays information on these ports in the current design. Each element in this list is either a collection of ports or a pattern matching the port names.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Displays information about ports in the current design. By default, the command produces a brief report including all ports in the design.

The input and output delay information lists the minimum rise, minimum fall, maximum rise, maximum fall delays, and the clock to which the delay is relative. If "(f)" follows the clock name, the delay is relative to the falling edge of the clock; otherwise the delay is relative to the rising edge. If "(l)" follows the clock name, input delay is considered to be coming from a level-sensitive latch, or output delay is considered to be going to a level-sensitive latch. By default, the delay is relative to a rising edge-triggered device.

Some information, such as whether the port is in an ideal network or not, is displayed after a timing update (as a result of manually executing an update_timing function or experiencing an implicit timing update as a result of executing other commands). This behavior is intended to help you to obtain information and statistics about ports without incurring the cost of a complete timing update.

EXAMPLES

This example shows the default port report.

```
ptc_shell> report_port

****************************************
Report : port
Design : middle
Version: ...
Date   : Wed Mar  8 07:26:43 2006
****************************************

Attributes:
    I - ideal network

                        Pin      Wire
Port           Dir      Cap      Cap     Attributes
-------------------------------------------------------
CLOCK          in       0.0000   0.0000
OPER           in       0.0000   0.0000
in0            in       0.0000   0.0000
in1            in       0.0000   0.0000
in2            in       0.0000   0.0000
out_1          out      0.0000   0.0000
out_2          out      0.0000   0.0000
out_3          out      0.0000   0.0000
out_4          out      0.0000   0.0000
```

```
p_in            in      0.0000   0.0000
p_out           out     0.0000   0.0000

1
```

SEE ALSO

report_design(2)
report_net(2)
report_cell(2)
report_reference(2)

report_reference

Reports the references in current instance or design.

SYNTAX

```
string report_reference [-nosplit]
```

ARGUMENTS

-nosplit

Does not split lines if the column overflows.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Displays information about all references in the current instance or current design. If the current instance is set, the report is generated relative to that instance; otherwise, the report is generated for the **current_design**.

EXAMPLES

The following is an example of **report_reference**.

```
ptc_shell> report_reference

****************************************
Report : reference
Design : middle
```

```
****************************************


Attributes:
    b - black box (unknown)
    h - hierarchical
    n - noncombinational

Reference       Library       Unit Area  Count  Total Area  Attributes
--------------------------------------------------------------------------------
FD2                tech_lib         3.00     4          12.00      b
ND2                tech_lib         3.00     4          12.00      b
inter                               6.00     2          12.00      h
low                                 2.00     2           4.00      h
--------------------------------------------------------------------------------
Total 4 references                                     40.00
```

SEE ALSO

report_hierarchy(2)
report_cell(2)

report_rule

Displays information about defined rules.

SYNTAX

```
string report_rule
   [rule_list]
   [-parameters]
   [-properties]
   [-types type_list]

list rule_list
list type_list
```

ARGUMENTS

rule_list

Displays information on these rules or rulesets. Each element in this list is either a collection of rules or rulesets, or a pattern matching the rule or ruleset names.

-parameters

Report parameters (if any) of the rules, and the value types accepted by each parameter.

-properties

Report properties of the rules (if applicable)

-types *type_list*

Specifies that the output of the report should only contain the corresponding rules of the given rule types and not all the rules available. The allowed values are analyze_design, block2top and sdc2sdc. The analyze_design value includes all the rules checked in analyze_design command. The block2top values include all the block to top rules that are checked in compare_block_to_top command. The sdc2sdc value includes all the constraint comparision rules checked in compare_constraints command.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Displays information about rules defined in this session. By default, the command produces a report including all defined rules. User-defined checker procedures associated with user-defined rules are displayed in a separate table.

EXAMPLES

This example shows the report for all rules matching the pattern "NTL_*".

```
ptc_shell> report_rule NTL_*

****************************************
Report : report_rules
Version: ...
Date   : Wed Aug 13 16:41:02 2008
****************************************

Rule          Severity  Status    Message
-------------------------------------------------------------------------------
NTL_0001      warning   enabled   Output port 'port' is unbuffered.
NTL_0002      error     enabled   Net 'net' has both strong and three-state drivers.
NTL_0003      warning   enabled   Net 'net' has potentially invalid multiple strong drivers.
NTL_0004      info      enabled   Net 'net' is a top-level feedthrough.
NTL_0005      warning   enabled   Unresolved reference to 'reference_name'.
1
```

SEE ALSO

analyze_design(2)
create_rule(2)
get_rules(2)

report_sense

Reports user specified unatenes and sense propagation constraints for data or clock.

SYNTAX

```
string report_sense
   [-type type]
   [-clocks clock_list]
   [-nosplit]
   object_list
```

Data Types

```
   clock_list       list
   object_list      list
```

ARGUMENTS

-type *type*

Specifies whether the type of sense being applied refers to clock networks or data networks. The possible values for the *type* variable are: 'clock', 'data'. Note that 'clock' is assumed to be the default if *-type* option is not given.

-clocks *clock_list*

Specifies a list of clock objects for which the report is generated. If the *-clocks* option is not supplied, all clocks passing through the given pin or arc objects are reported.

object_list

Lists of ports, pins or cell timing arcs to report.

-nosplit

Specifies not to split lines if a column overflows.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command reports the user-defined constraints applied to the senses and unateness propagation of data and clock signals.

EXAMPLES

The following example specifies and reports a stop sense propagation through an arc.

```
ptc_shell> set_sense -stop_propagation [get_timing_arcs -from u2/A -to u2/Z] \
           -clock [get_clock clk12]
ptc_shell> report_sense -type clock
****************************************
Report : sense
         -type clock
Design : simple_2ff
Version: G-2012.06
Date   : Mon Apr  9 13:50:15 2012
****************************************


Object                          Clock               Sense
------------------------------------------------------------------------------
u2/A -> Z                       clk12               stop_prop
1
```

The following example selects positive unateness for a pin named **MUX1/Z** for all clocks.

```
ptc_shell> set_sense  -positive MUX1/Z
ptc_shell> report_sense
****************************************
Report : sense
         -type clock
Design : nonunate
Version: G-2012.06-alpha-DEV
Date   : Mon Apr  9 13:50:15 2012
****************************************


Object                          Clock            Sense
------------------------------------------------------------------------------
MUX1/Z                          *                positive
```

SEE ALSO

```
set_sense(2)
remove_sense(2)
```

report_transitive_fanin

Reports logic in fanin of specified sink objects.

SYNTAX

```
string report_transitive_fanin [-nosplit]
   -to sink_list
   [-trace_arcs arc_types]

list sink_list
```

ARGUMENTS

-nosplit

Does not split lines if column overflows.

-to *sink_list*

Specifies a list of sink pins, ports, or nets in the design, whose transitive fan-in is reported. If a net is specified, the effect is the same as listing all driver pins on the net.

-trace_arcs arc_types

Specifies the type of combinational arcs to trace during the traversal. Allowed values are **timing** (the default), which permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis); **enabled**, which permits the tracing of all enabled arcs and disregards case analysis values; and **all**, which permits the tracing of all combinational arcs regardless of either case analysis or arc disabling.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The command produces a report showing the transitive fanin of specified pins, ports, or nets in the design. A pin is considered to be in the transitive fanin of a sink if there is a timing path through combinational

logic from the pin to that sink (please also see the **-trace_arcs** option). The fanin report stops at the clock pins of registers (sequential cells).

If the **current_instance** is set, the report will be focussed on the fanin within the **current_instance**. The report stops at the boundaries of the **current_instance**, and no paths outside the **current_instance** are reported. The report shows the hierarchical boundary pins on the current instance.

EXAMPLES

The following example shows the transitive fanin of a pin in the design.

```
ptc_shell>report_transitive_fanin -to FF1/D

****************************************
Report : transitive_fanin
Design : top
Version: ...
Date   : Tue Aug  6 14:53:47 1996
****************************************

Fanin network of sink 'FF1/D':


Driver Pin       Load Pin          Type        Sense
-------------------------------------------------------------
gate1/Y          FF1/D             (net arc)   same


Load Pin         Driver Pin        Type        Sense
-------------------------------------------------------------
gate1/A          gate1/Y           AN2         same
gate1/B          gate1/Y           AN2         same


Driver Pin       Load Pin          Type        Sense
-------------------------------------------------------------
IN1              gate1/A           (net arc)   same
IN2              gate1/B           (net arc)   same
```

SEE ALSO

current_design(2)
current_instance(2)
report_clock(2)
report_transitive_fanout(2)

report_transitive_fanout

Reports logic in fanout of specified sources.

SYNTAX

```
string report_transitive_fanout [-nosplit]
   -clock_tree
   -from source_list
   [-trace_arcs arc_types]

list source_list
```

ARGUMENTS

-nosplit

Does not split lines if column overflows.

-clock_tree

Reports transitive fanout of all clock sources in the design.

-from *source_list*

Specifies a list of source pins, ports, and nets in the design whose transitive fanout is reported. If a net is specified, the effect is same as listing all the load pins on the net.

-trace_arcs arc_types

Specifies the type of combinational arcs to trace during the traversal. Allowed values are **timing** (the default), which permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis); **enabled**, which permits the tracing of all enabled arcs and disregards case analysis values; and **all**, which permits the tracing of all combinational arcs regardless of either case analysis or arc disabling.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Produces a report showing the transitive fanout of specified pins or ports in the design. A pin is considered to be in the transitive fanout of a source if there is a timing path through combinational logic from the source to that pin (please also see the **-trace_arcs** option). The fanout report stops at the inputs to registers (sequential cells).

If the **current_instance** is set, the report focuses on the fanout within the **current_instance**. The report stops at the boundaries of the **current_instance**, and no paths outside the **current_instance** are reported. The report also shows the hierarchical boundary pins on the **current_instance**.

EXAMPLES

The following example shows the transitive fanout of a pin in the design.

```
ptc_shell> report_transitive_fanout -from FF1/Q

****************************************
Report : transitive_fanout
Design : top
Version: ...
Date   : Tue Aug  6 15:19:19 1996
****************************************

Fanout network of source 'FF1/Q':


Driver Pin       Load Pin          Type        Sense
------------------------------------------------------------
FF1/Q            gate5/A           (net arc)   same


Load Pin         Driver Pin        Type        Sense
------------------------------------------------------------
gate5/A          gate5/Y           AN2         same


Driver Pin       Load Pin          Type        Sense
------------------------------------------------------------
gate5/Y          OUT2              (net arc)   same
```

The following command shows the transitive fanout of all the clock sources in the design.

```
ptc_shell> report_transitive_fanout -clock_tree

****************************************
Report : transitive_fanout
        -clock_tree
Design : top
Version: ...
Date   : Tue Aug  6 15:24:00 1996
****************************************

Fanout network of source 'CLK':
```

```
Driver Pin       Load Pin          Type        Sense
-------------------------------------------------------------
CLK              FF3/CLK           (net arc)   same
CLK              FF2/CLK           (net arc)   same
CLK              FF1/CLK           (net arc)   same
```

SEE ALSO

create_clock(2)
current_design(2)
current_instance(2)
report_clock(2)
report_transitive_fanin(2)

report_units

Reports the unit information for the current design.

SYNTAX

```
string report_units
   [-nosplit]
```

ARGUMENTS

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **report_units** command displays the units used by the current_design. To generate the report, the design should be loaded and linked to a library. The units are always reported in reference to MKS units like Farads, Amps, Ohms, Seconds, Volts, and Watts. The resistance unit is inferred from the time and capacitance units. Both time and capacitance units are required for constraint consistency. If either of them is missing the unit is undefined and it is reported as "Not specified". The most common cause of this problem is that the library is in very old format, and it needs to be recompiled with the latest **Library Compiler**.

EXAMPLES

```
ptc_shell> report_units
```

****
Report : report_units
Design : .
Version: .
Date : .
****
Input Units (set by technology library)
---
time : 1.00ns
resistance : 10.00MOhm
capacitance : 100.00aF
voltage : 1.00V
current : 1.00uA
leakage_power : 1.00mW
Output Units (set by technology library)
---
time : 1.00ns
resistance : 10.00MOhm
capacitance : 100.00aF
voltage : 1.00V
current : 1.00uA
leakage_power : 1.00mW

SEE ALSO

read_db(2)
set_min_library(2)

report_waiver

Displays information about existing waivers.

SYNTAX

```
Boolean report_waiver
   [-names waiver_name_list]
   [-rules rule_name_list]
   [-design design]
   [-scenarios scenario_name_list]

list waiver_name_list
list rule_name_list
string design
list scenario_name_list
```

ARGUMENTS

-names *waiver_name_list*

Report the waivers of the given names only.

-rules *rule_name_list*

Report the waivers for the given rules only.

-design *design*

Report the waivers for the given design only.

-scenarios *scenario_name_list*

Report the waivers active in the given scenarios only. This option should be used in conjuction of the "-design" option.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Displays information about existing violation waivers that meet the given criteria. If none of *-names*, *-rules*, *-design* or *-scenarios* is given, all active waivers will be included in the report.

EXAMPLES

The following example reports all waivers for the design.

```
ptc_shell> report_waiver

*************************************
Report : report_waiver
Version: C-2009.06-SP2
Date   : Mon Aug 17 14:47:52 2009
*************************************

Name          Rule      Scenario  Condition                      Comment
-------------------------------------------------------------------------------
my_waiver1    CLK_0026  default   -condition [list "clock" [get_clocks {clk1 clk2}]]
                                                                 "test 1"
my_waiver2    CLK_0003  default   -condition [list "clock" [get_clocks {clk1}]
                                  -not_condition [list "pin" [get_pins U1/A]]
                                                                 "test 2"

waiver_0                default   -cells {U1/U1}                 "test 3"
```

SEE ALSO

analyze_design(2)
create_waiver(2)
remove_waiver(2)
write_waiver(2)

reset_design

Removes user specified information from design.

SYNTAX

```
string reset_design [-timing]
```

ARGUMENTS

-timing

Resets the timing information on the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Removes all attributes from the design. Whether these attibutes were created as user attributes, were created as a result of commands, or were read from design db does not make any difference. Attributes include but not limited to exceptions, input/output delays, clocks, etc.

EXAMPLES

The following command resets the attributes on the current design.

ptc_shell> **reset_design**

SEE ALSO

current_design(2)
remove_user_attribute(2)

reset_mode

Resets cell mode groups to the default state.

SYNTAX

```
string reset_mode
   [-type cell | design]
   [-group group_list]
   [instance_list]
```

Data Types

```
   group_list           list
   instance_list        list
```

ARGUMENTS

-type cell | design

Indicates the type of mode group to be set to default. This option has the following mutually-exclusive valid values: *design* and *cell*. Note that design modes are now obsolete. The *design* option is kept for backward compatibility.

- The *cell* value specifies that cell mode groups are to be set to default. Cell mode groups are defined on library cells in the library.

-group *group_list*

Specifies a list of cell mode groups, each of which is to be reset to its default setting.

instance_list

Specifies a list of instances for which the specified cell mode groups are to be set to default. If no instance list is specified, all mode cells are considered. No error occurs if you reset the modes on a cell that has no modes.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Resets cell mode groups to the default state of all modes enabled. Cell mode groups are defined in the library. Each library cell can have a set of cell mode groups. Each of these cell mode groups can have two or more cell modes. Each of these cell modes can be mapped to a set of timing arcs of the library cell.

When a cell mode group is set to default for a given instance of the library cell all of its modes are enabled for that cell. All timing arcs of the enabled modes also get enabled with respect to modes for that cell.

Cell mode groups can have different states due to the **set_mode -type cell**, and conditional evaluation. When conflict arises the following precedence rule is employed:

- **set_mode -type cell**
- Evaluation of mode conditions

To reset the state of the cell mode group due to the **set_mode -type cell** command, use the **reset_mode** command.

To reset the state of the cell mode group due to conditional evaluation, use the **remove_case_analysis** command or edit the Verilog netlist to remove logical constants.

EXAMPLES

In the following example, two cell mode groups, RW and SH, are defined on cell Uram1/D0. The first command sets READ as the active mode for RW and EARLY as the active mode for SH.

```
ptc_shell> set_mode {EARLY READ} Uram1/D0
```

The following command can be employed to reset the cell mode groups to default.

```
ptc_shell> reset_mode -group {RW SH} Uram1/D0
```

To reset all cell mode group in Uram1/DO use the following command

```
ptc_shell> reset_mode Uram1/D0
```

To reset all cell mode groups in the design use

```
ptc_shell> reset_mode
```

SEE ALSO

```
report_mode(2)
set_mode(2)
set_case_analysis(2)
```

reset_path

Resets specified paths to single-cycle behavior.

SYNTAX

```
Boolean reset_path
   [-setup] [-hold]
   [-rise] [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-all]

list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
```

ARGUMENTS

-setup

Indicates that only setup (maximum delay) evaluation is to be reset to its default, single-cycle behavior. If neither **-setup** nor **-hold** is specified, both setup and hold checking are reset to single-cycle.

-hold

Indicates that only hold (minimum delay) evaluation is to be reset to its default, single-cycle behavior. If neither **-setup** nor **-hold** is specified, both setup and hold checking are reset to single-cycle.

-rise

Indicates that only rising path delays are to be reset to single-cycle behavior. If neither **-rise** nor **-fall** is specified, both rising and falling delays are reset to single-cycle.

-fall

Indicates that only falling path delays are to be reset to single-cycle behavior. If neither **-rise** nor **-fall** is specified, both rising and falling delays are reset to single-cycle.

-all

A shortcut to reset all the paths to single-cycle behavior.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified.If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, and nets through which the paths must pass that are to be reset. Nets are interpreted to imply the net segment's local driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than once in one command invocation. For a discussion of the use of multiple **-through** options, see the DESCRIPTION section.

-rise_through ***rise_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the through points. You can specify **-rise_through** more than once in one command invocation. For a discussion of the use of multiple **-through** options, see the DESCRIPTION section.

-fall_through ***fall_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the through points. You can specify **-fall_through** more than once in one command invocation. For a discussion of the use of multiple **-through** options, see the DESCRIPTION section.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a

clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The command restores designated timing paths in the current design the default single-cycle behavior. Use **reset_path** to undo the effect of **set_multicycle_path**, **set_false_path**, **set_max_delay**, and **set_min_delay**. Attributes set by these commands are removed by **reset_path**.

Note that a general **reset_path** command removes the effect of matching general or specific point-to-point exception commands, as long as there is a matching object in the path specification of the original exception-setting commands. For example, the following command

```
reset_path -from {CLK}
```

resets more specific matching exceptions for the object CLK, such as

```
set_false_path -from {CLK} -to {d/Z}
set_false_path -from {CLK} -to {g/Z}
```

If you do not specify either **-setup** or **-hold**, the default behavior is restored to both. The default is a setup relation of one cycle and a hold relation of zero cycles, so that hold is checked one active edge before the setup data at the destination register.

Setup and hold information is different for rising and falling transitions. In most cases, these are reset together. Use the **-rise** or **-fall** options to reset only one or the other. To disable setup or hold calculations for paths, use **set_false_path**.

The general and specific constraint options (for example, **-through** and **-rise_through** or **-fall_through**) are treated as different qualifiers. That is, if you issue **reset_path -fall_through**, only those constraints are removed that were set with the **-fall_through** option; any that were set with the **-through** option are not removed. The same applies to the general and specific **-to** and **-from** options. For an example, see the EXAMPLES section.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to

specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

In some earlier releases of constraint consistency, a single **-through** option specifies multiple traversal points. You can revert to this behavior by setting the **timing_through_compatability** variable to true. If you do so, the behavior is as illustrated in the following example, which specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through {B1 C1} -to D1
```

In this mode, you cannot use more than one **-through** option in a single command.

EXAMPLES

The following command resets the timing relation between ff1/CP and ff2/D to the default single-cycle behavior.

ptc_shell> **reset_path -from { ff1/CP } -to { ff2/D }**

The following command resets only the rising delay to single cycle for all paths ending at latch2d.

ptc_shell> **reset_path -rise -to latch2d**

The following example first sets a specific and a general point-to-point exception, then removes the exceptions.

```
ptc_shell> set_multicycle_path 2 -from A -to Z
ptc_shell> set_max_delay 15 -to Z
ptc_shell> reset_path -to Z
```

In following example, the **reset_path** command removes only the max_delay constraints, because **-to** and **-rise_to** are treated as different qualifiers and do not match.

```
ptc_shell> set_multicycle_path 2 -from A -to Z
ptc_shell> set_max_delay 15 -rise_to Z
ptc_shell> reset_path -rise_to Z
```

SEE ALSO

current_design(2)
reset_design(2)
set_false_path(2)
set_max_delay(2)
set_min_delay(2)
set_multicycle_path(2)

reset_timing_derate

Resets user-specified derate factors set either on a design or on a specified list of instances, such as cells, nets or library cells.

SYNTAX

```
string reset_timing_derate
```

ARGUMENTS

object_list

Specifies current design or a list of cells, library cells, or nets which are reset.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Call this command with no arguments to reset all derate factors set on the design (to the default value, 1.0). This includes both derate factors set globally on the design and those set on specific instances in the design. Call this command with a list of objects to reset a specific set of objects.

The *object_list* option may be used to reset derate factors on specific objects (cells, library cells, or nets) in the design. All derate factors are reset on each instance. If the *object_list* option contains a design, then only global derate factors are reset and instance-specific derate factors are not reset.

Multicorner-Multimode Support

This command uses information from the current scenario only.

EXAMPLES

The following example removes derates for all of the objects in the current design or libraries.

```
ptc_shell> reset_timing_derate
```

SEE ALSO

set_timing_derate(2)

restore_session

Restores a constraint consistency session from a directory saved by the **save_session** command.

SYNTAX

```
int restore_session
   directory_name
```

Data Types

```
   directory_name     string
```

ARGUMENTS

directory_name

Specifies the name of a directory to read the session information from.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Use this command to read a directory that was written by the **save_session** command and restore a constraint consistency session to the same state as the session where the **save_session** command was issued. Note that if there are any current designs or libraries loaded before executing this command, they are removed.

The design data is restored from data in the session directory.

When a saved image is restored, any variable that existed in the saved image is restored to its saved value. However, any variables that exist in an analysis before the **restore_session** command is issued, which were not present in the saved image, remain unchanged.

Note that the same version of the tool has to be used to restore a session stored with a specific version of the tool.

EXAMPLES

This example reads a constraint consistency savable image in a directory named state1.

```
ptc_shell> restore_session state1
```

SEE ALSO

save_session(2)

save_session

Saves data of a constraint consistency session in the named directory so that it can be restored later using the **restore_session** command.

SYNTAX

```
int save_session
   dir_name
```

Data Types

```
   dir_name      string
```

ARGUMENTS

dir_name

Specifies the name of a directory to save the session in. If the named directory does not exist, the **save_session** command tries to create it. If it already exists, the **save_session** command issues an information message and overwrites the directory with the new session.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Use this command to create a repository of data to save the current constraint consistency session to. The name of the directory to save the session is a required argument. If the directory does not exist, a new one is created and the data for the session is written into the directory. If the directory exists, an information message is issued and the directory replaced by the data of the current session to be saved.

Note: There are a few things that are not saved. These include: collections that involve non-netlist objects, Tcl procedures, and message (warning/information) suppressions.

Note: Unlike PrimeTime, constraint consistency is self sufficient and does not require rereading of technology libraries. In addition, constraint consistency can save and restore all designs in memory (not

just the current design), including all of the linked designs (note there can be more than one linked design).

EXAMPLES

This example saves a constraint consistency session to a directory named state1:

```
ptc_shell> save_session state1
```

SEE ALSO

restore_session(2)

set_annotated_transition

Sets the transition time annotated on specified pins in the current design.

SYNTAX

```
int set_annotated_transition
   [-rise]
   [-fall]
   [-min]
   [-max]
   [-delta_only]
   slew_value
   pin_list
```

Data Types

slew_value	float
pin_list	list

ARGUMENTS

-rise

Indicates that the *slew_value* variable represents the data rise transition time.

-fall

Indicates that the *slew_value* variable represents the data fall transition time.

-min

Indicates that the *slew_value* variable represents the minimum transition time. Use this option only if the design is in min-max mode (min and max operating conditions).

-max

Indicates that the *slew_value* variable represents the maximum transition time. Use this option only if the design is in min-max mode (min and max operating conditions).

-delta_only

Indicates that the *slew_value* variable represents a delta transition time that is added to the transition

time computed by delay calculation.

slew_value

Specifies the transition time of the specified pins or ports, in units consistent with the technology library used during optimization. For example, if the technology library specifies delay values in nanoseconds, the *slew_value* variable must be expressed in nanoseconds. If used with the *-delta_only* option, the *slew_value* variable can be a negative number.

pin_list

Specifies a list of pins or ports that is annotated with the transition time *slew_value*.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies the transition time that is annotated on pins in the current design. The specified transition time value overrides the internally-estimated transition time value.

The **set_annotated_transition** command can be used for pins at lower levels of the design hierarchy. Pins are specified as "INSTANCE1/INSTANCE2/PIN_NAME."

To remove annotated transition times, use the **remove_annotated_transition** command.

EXAMPLES

The following example annotates a rising transition time of 0.5 units and a falling transition time of 0.7 units on input pin **A** of cell "U1/U2/U3".

```
ptc_shell> set_annotated_transition -rise 0.5 [get_pins U1/U2/U3/A]
ptc_shell> set_annotated_transition -fall 0.7 [get_pins U1/U2/U3/A]
```

SEE ALSO

```
remove_annotated_transition(2)
reset_design(2)
```

set_app_var

Sets the value of an application variable.

SYNTAX

```
string set_app_var
   -default
   var
   value
```

Data Types

```
   var        string
   value      string
```

ARGUMENTS

-default

Resets the variable to its default value.

var

Specifies the application variable to set.

value

Specifies the value to which the variable is to be set.

DESCRIPTION

The **set_app_var** command sets the specified application variable. This command sets the variable to its default value or to a new value you specify.

This command returns the new value of the variable if setting the variable was successful. If the application variable could not be set, then an error is returned indicating the reason for the failure.

Reasons for failure include:

- The specified variable name is not an application variable, unless the application variable **sh_allow_tcl_with_set_app_var** is set to true See the **sh_allow_tcl_with_set_app_var** man page for details.
- The specified application variable is read only.
- The value specified is not a legal value for this application variable

EXAMPLES

The following example attempts to set a read-only application variable:

```
prompt> set_app_var synopsys_root /tmp
Error: can't set "synopsys_root": variable is read-only
       Use error_info for more info. (CMD-013)
```

In this example, the application variable name is entered incorrectly, which generates an error message:

```
prompt> set_app_var sh_enabel_page_mode 1
Error: "sh_enabel_page_mode" is not an application variable
        Use error_info for more info. (CMD-013)
```

This example shows the variable name entered correctly:

```
prompt> set_app_var sh_enable_page_mode 1
1
```

This example resets the variable to its default value:

```
prompt> set_app_var sh_enable_page_mode -default
0
```

SEE ALSO

```
get_app_var(2)
report_app_var(2)
write_app_var(2)
```

set_case_analysis

Specifies that a port or pin is at a constant logic value 1 or 0, or is considered with a rising or falling transition.

SYNTAX

```
string set_case_analysis value
   port_or_pin_list

string 0 | 1 | rising | falling
list port_or_pin_list
```

ARGUMENTS

value

Specifies a constant logic value or a transition to assign to the given pin or port. The valid constant values are **0**, **1**, **zero**, and **one**. The valid transition values are **rising**, **falling**, **rise**, and **fall**.

port_or_pin_list

Lists ports or pins to which the case analysis is assigned. In the case of pins, constant propagation is executed forward only. No backward constant propagation is performed.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Case analysis is a way of specifying a given mode for the design without altering the netlist structure. You can specify for the current timing analysis session, that some signals are at a constant value (**1** or **0**), or that only one type of transition (**rising** or **falling**) is considered.

Pins of a design may be driven by logic values from the design, but if case analysis is used to set a value on such pins, the values set by case analysis will dominate. If the case values are subsequently removed, the value driving the pin from the design will be propagated.

When case analysis is specified as a constant value, this value is propagated through the network as long as the constant value is a controlling value for the traversed logic. For example, if you specify that one of the inputs of a NAND gate is a constant value **0**, it is propagated to the NAND output, which is now considered at a logic constant **1**. This propagated constant value is then propagated to all cells driven by this signal.

Note: When a logic value is propagated onto a a net, the associated DRC constraint checks will be disabled. However max_capacitance DRC check can be performed on driver pins for constant nets by setting variable **timing_enable_max_capacitance_set_case_analysis** to **TRUE**. This variable is set to **FALSE** by default.

In the event that the case analysis value is a transition, the given pin or port is considered only for timing analysis with the specified transition. The other transition is disabled. The case analysis information is used by all analysis commands.

Case analysis is used in addition to the mode commands to fully specify the mode of a design. For example, a design that instantiates models with a TESTMODE, is specified so that the TESTMODE is disabled during the timing analysis session by using the **set_mode** command. In addition, if a TESTMODE signal exists on the design, it is specified to a constant logic value, so that all test logic controlled by the TESTMODE signal is disabled.

The -rising and -falling options are ignored constraint consistency during power calculation.

EXAMPLES

The following example shows that the port *IN1* is at a constant logic value of **0**.

```
ptc_shell> set_case_analysis 0 IN1
```

The following example shows that the pins *U1/U2/A* and *U1/U3/CI* are considered only for a rising transition. The falling transition on these pins is disabled.

```
ptc_shell> set_case_analysis rising {U1/U2/A U1/U3/CI}
```

The following example specifies how to disable the TESTMODE of a design for which instances are models having a TESTMODE mode. The design also has a *TEST_PORT* port set to a constant logic value **0**.

```
ptc_shell> remove_mode TESTMODE U1/U2
ptc_shell> set_case_analysis 0 TEST_PORT
```

SEE ALSO

remove_case_analysis(2)
report_case_analysis(2)
disable_case_analysis(3)
set_mode(2)

set_clock_gating_check

Specifies the value of setup and hold time for clock gating checks.

SYNTAX

```
string set_clock_gating_check
   [-setup setup_value]
   [-hold hold_value]
   [-rise | -fall]
   [-high | -low]
   [object_list]

float setup_value
float hold_value
list object_list
```

ARGUMENTS

-setup *setup_value*

Specifies the clock gating setup time. The default is 0.0.

-hold *hold_value*

Specifies the clock gating hold time. The default is 0.0.

-rise

Indicates that only rising delays are to be constrained. By default, if neither **-rise** nor **-fall** are specified, both rising and falling delays are constrained.

-fall

Indicates that only falling delays are to be constrained. By default, if neither **-rise** nor **-fall** are specified, both rising and falling delays are constrained.

-high

Indicate that the check is to be performed on the high level of the clock. By default, constraint consistency determines whether to use the high or low level of the clock using information from the cell's logic. That is, for AND and NAND gates constraint consistency performs the check on the high level; for OR and NOR gates, on the low level. For some complex cells (for example, MUX, OR-AND)

constraint consistency cannot determine which to use, and does not perform checks unless you specify either **-high** or **-low**. If the user-specified value differs from that derived by constraint consistency, the user-specified value takes precedence, and a warning message is issued. If you specify **-high** or **-low** you must also specify *object_list*; in that case, *object_list* must not contain a clock or a port.

-low

Indicates that the check is to be performed on the low level of the clock. By default, constraint consistency determines whether to use the high or low clock level using information from the cell's logic. That is, for AND and NAND gates constraint consistency performs the check on the high level; for OR and NOR gates, on the low edge. For some complex cells (for example, MUX, OR-AND) constraint consistency cannot determine which to use, and does not perform checks unless you specify either **-high** or **-low**. If the user-specified value differs from that derived by constraint consistency, the user-specified value takes precedence, and a warning message is issued. If you specify **-high** or **-low** you must also specify *object_list*; in that case, *object_list* must not contain a clock or a port.

object_list

Specifies a list of objects in the current design for which the clock gating check is to be applied. The objects can be clocks, ports, pins, or cells. If a cell is specified, all input pins of that cell are affected. If a pin, cell, or port is specified, all gates in the transitive fanout are affected. If a clock is specified, the clock gating check is applied to all gating gates driven by that clock. If you specify **-high** or **-low** you must also specify *object_list*; in that case, *object_list* must not contain a clock or a port. By default, if *object_list* is not specified, the clock gating check is applied to the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_clock_gating_check** command specifies a setup or hold time clock gating check to be used for clocks, ports, pins, or cells. The gating check is performed on pins that gate a clock signal.

The clock gating setup check is used to ensure the controlling data signals are stable before the clock is active. This check is performed on combinational gates through which the clock signals are propagated. The arrival time of the leading edge of the clock pin is checked against both levels of any data signals gating the clock. A clock gating setup failure can cause either a glitch at the leading edge of the clock pulse, or a clipped clock pulse.

The clock gating hold check is used to ensure that the controlling data signals are stable while the clock is active. The arrival time of the trailing edge of the clock pin is checked against both levels of any data signal gating the clipped clock pulse.

The options **-high** and **-low** are intended to specify clock gating checks that constraint consistency cannot determine. These options apply only to the pins specified and not to the cells or pins in the transitive fanout. Conversely, setup and hold values for clock gating checks apply to the cells and pins in the transitive fanout if the variable **timing_clock_gating_check_fanout_compatibility** is set. If you need to use the **-setup** or **-hold** options and also the **-high** or **-low** options, issue two separate **set_clock_gating_check** commands to avoid confusion as to what is propagated and what is not propagated if the variable **timing_clock_gating_check_fanout_compatibility** is set.

Use the **remove_clock_gating_check** command to remove information set by

set_clock_gating_check. The **reset_design** command removes all attributes from the design, including those set by **set_clock_gating_check**.

Use the **report_clock_gating_check** command to report the information for the clock gating check.

set_clock_gating_check does not alter clock gating checks that come directly from the library. These checks are marked as "library defined" by the command **report_clock_gating_check**. Library defined checks can only be altered with the command set_annotated_check or with SDF annotations.

EXAMPLES

The following example specifies a setup time of 0.2 and a hold time of 0.4 for all gates in the clock network of clock CK1.

```
ptc_shell> set_clock_gating_check -setup 0.2 -hold 0.4 [get_clocks CK1]
```

The following example specifies a setup time of 0.5 on the gate and1.

```
ptc_shell> set_clock_gating_check -setup 0.5 [get_cells and1]
```

SEE ALSO

remove_clock_gating_check(2)
current_design(2)
reset_design(2)
report_clock_gating_check(2)

set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during the timing analysis.

SYNTAX

```
Boolean set_clock_groups
   [-physically_exclusive | -logically_exclusive | -asynchronous]
   [-allow_paths]
   [-name name]
   [-comment comment_string]
   [-group clock_list]*

list clock_list
string  comment_string
```

ARGUMENTS

Note: Options marked with asterisks (*) in the SYNTAX can be used more than once in the same command.

-physically_exclusive

Specifies that the clock groups are physically exclusive with each other. Physically exclusive clocks cannot co-exist in the design physically. An example of this is multiple clocks that are defined on the same source pin. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-logically_exclusive

Logically exclusive clocks do not have any functional paths between them, but may have coupling interactions with each other. An example of logically-exclusive clocks is multiple clocks, which are selected by a MUX but can still interact through coupling upstream of the MUX cell. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-asynchronous

Specifies that the clock groups are asynchronous to each other. Two clocks are asynchronous with respect to each other if they have no phase relationship at all. Signal integrity analysis uses an infinite arrival window on the aggressor unless all the arrival windows on the victim net and the aggressor net are defined by synchronous clocks. The **-physically_exclusive**, **-logically_exclusive** and **-**

asynchronous options are mutually exclusive; you must choose only one.

-allow_paths

Enable the timing analysis between specified asynchronous clock groups. The default behavior is to suppress timing paths between asynchronous clocks. The *-allow_paths* option must be used with *-asynchronous* option.

-name ***name***

Specifies a name for the clock grouping to be created. Each command should specify a unique name, which identifies the exclusive or asynchronous relationship among specified clock groups, and this name is used later on to easily remove the clock grouping defined here. By default, the command creates a unique name.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

-group ***clock_list***

Specifies a list of clocks. You can use the **-group** option more than once in a single command execution. Each **-group** iteration specifies a group of clocks, which are exclusive or asynchronous with the clocks in all other groups. If only one group is specified, it means that the clocks in that group are exclusive or asynchronous with all other clocks in the design. Thus, a default other group is created for this single group. Whenever a new clock is created, it is automatically included in the default exclusive or asynchronous group. Substitute the list you want for *clock_list*.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during the timing analysis. This is similar to using the set_false_path command. In addition, these relationships also imply the type of crosstalk analysis which should be performed between the clocks. A clock cannot be present in multiple groups during one **set_clock_groups** command execution, but can be included in multiple groups by executing multiple **set_clock_groups** commands. Clock relationships are only implied across the specified clock groups; no relationship is implied across clocks within a group.

For all three relationships, timing paths between the clocks are suppressed. The relationships differ in how crosstalk is handled. If clocks are asynchronous, the clocks are assumed to have an infinite window relationship with each other. If clocks are physically exclusive, only one clock can act as an aggressor or victim during crosstalk analysis; interactions between the clocks will not be explored. If clocks are logically exclusive, the crosstalk computation will be computed normally (synchronously if the clocks are synchronous) even though the timing paths between the clocks are not reported.

Multiple relationship types can exist between two clocks. When this happens, the relationships obey the following precedence:

```
* physically exclusive (highest)
* asynchronous
* logically exclusive (lowest)
```

These precedence rules reflect the real physical behavior of the resulting circuit. For example, if two clocks are asynchronous but they are never simultaneously present, then no crosstalk should be computed.

Note that the traditional set_false_path exception between clocks will not result in infinite window crosstalk computation between two clocks. If multiple clocks are asynchronous with each other, the asynchronous relationship should be specified with set_clock_groups. Failure to specify this relationship for asynchronous clocks could result in an optimistic analysis!

A special -allow_paths modifier can be used with the -asynchronous option. When used, the clocks are assumed to have an infinite window relationship for crosstalk purposes, but timing paths between the clocks will be treated normally. When using this option, paths between the clock domains can be manually disabled using the set_false_path command.

When a clock pair is defined such that it has false paths (using either physically or logically exclusive or asynchronous) but subsequently modified to have -asynchronous -allow_paths, then this is an error and the first definition holds. In the reverse situation, also the redefinition to false paths from -allow-paths between the clock pairs would also be an error. Please refer message UITE-457 and UITE-458.

This command should be used to specify the relationships among clocks before using the **set_active_clocks** command. You should not also set a false path if you have already specified two clocks as exclusive or asynchronous. An error message is issued if you attempt to set a false path between two clocks which are already exclusive or asynchronous. If a false path was previously set between two clocks when an exclusive or asynchronous relationship is applied, the false path is overwritten by the **set_clock_groups** command. Other exceptions are unaffected.

A clock relationship on a master clock will not automatically propagate to any generated clocks. If the relationship should also apply to generated clocks, they should be explicitly included in the set_clock_groups command.

To undo the **set_clock_groups** command, use the **remove_clock_groups** command. To report the clock groups defined in a design, use the **report_clock** command with the **-groups** option.

EXAMPLES

The following example defines two asynchronous clock domains.

```
ptc_shell> set_clock_groups -asynchronous -name g1 -group CLK33 -group CLK50
```

The following example defines a clock named *TESTCLK* to be asynchronous with all other clocks in the design:

```
ptc_shell> set_clock_groups -asynchronous -group TESTCLK
```

The following example shows how to simultaneously analyze multiple clocks per register without manually specifying false paths. Assume two pairs of mutually-exclusive clocks that are multiplexed:

CLK1 and *CLK2*
CLK3 and *CLK4*.

If each pair of clocks is selected by a different signal, you must execute two **set_clock_groups** commands to specify two independent exclusivity relationships, one for each MUX selection signal:

```
ptc_shell> set_clock_groups -physically_exclusive -group CLK1 -group CLK2
ptc_shell> set_clock_groups -physically_exclusive -group CLK3 -group CLK4
```

If each pair of clocks is selected by a single MUX selection signal, you would execute only one **set_clock_groups** command to simultaneously analyze all four clocks.

```
ptc_shell> set_clock_groups -physically_exclusive -group {CLK1 CLK3} -group {CLK2 CLK4}
```

In this case, we are not implying any type of relationship between CLK1-CLK3 or CLK2-CLK4. For example, f CLK1 and CLK3 were also asynchronous with each other, we would need to specify an additional relationship between them.

SEE ALSO

remove_clock_groups(2)
report_clock(2)
set_active_clocks(2)
set_false_path(2)
create_clock(2)
create_generated_clock(2)

set_clock_jitter

Specifies the cycle and duty_cycle jitters of specified clocks.

SYNTAX

```
string set_clock_jitter
   jitter
   [-clock clock_list]
   [-cycle]
   [-duty_cycle]

list clock_list
float cycle
float duty_cycle
```

ARGUMENTS

-clock *clock_list*

Specifies a list of clocks on which to set the clock jitter.

-cycle *cycle_to_cycle_jitter*

Specifies the cycle-to-cycle jitter on the clock.

-duty_cycle *duty_cycle_jitter*

Specifies the duty-cycle jitter on the clock.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command specifies the cycle-to-cycle jitter and the duty-cycle jitter on a primary master clock. A primary master clock is a clock that is not a generated clock.

Cycle-to-cycle jitter models the variation between the edges that are in-phase. In other words, it models

the variation between the edges that are multiple clock cycles apart.

Duty-cycle jitter models the variation between the edges that are out-of-phase. In other words, they are 0.5, 1.5, 2.5, and so on, cycles apart.

To remove the clock jitters set by the **set_clock_jitter** command, use the **remove_clock_jitter** command.

EXAMPLES

The following example specifies a cycle-to-cycle jitter of 0.5 and duty-cycle jitter of 0.7 on the primary master clock mclk

```
ptc_shell> set_clock_jitter -clock [get_clocks mclk] -cycle 0.5 -duty_cycle 0.7
```

SEE ALSO

remove_clock_jitter(2)

set_clock_latency

Specifies latency of clock network.

SYNTAX

```
string set_clock_latency
   [-clock clock_list]
   [-rise][-fall]
   [-min][-max]
   [-source]
   [-late][-early]
   [-dynamic dynamic_component_of_delay]
   [-pll_shift]
   delay
   object_list

list clock_list
float delay
float   dynamic_component_of_delay
list object_list
```

ARGUMENTS

-clock *clock_list*

Specifies a list of clock objects to be associated with the network latency that is placed on all pin/port objects in object_list. If the **-clock** option is supplied when object_list refers to clock objects, a warning is issued that the option is not relevant in this case and execution of the command proceeds as if -clock was not given. A more specific clock latency setting overrides a more general one.

-rise

Specifies clock rise latency.

-fall

Specifies clock fall latency.

-min

Specifies clock latency for the minimum operating condition.

-max

Specifies clock latency for the maximum operating condition.

-source

Specifies clock source latency.

-late

Specifies clock late source latency.

-early

Specifies clock early source latency.

-dynamic ***dynamic_component_of_delay***

Specifies dynamic component of clock latency value.

-pll_shift

Specifies that latencies correspond to PLL shifts. This option applies only to PLL output clocks.

delay

Specifies clock latency value.

object_list

Lists clocks, ports, or pins.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Two types of clock latency can be specified for a design: The clock network latency and the clock source latency.

The clock network latency is the time it takes a clock signal to propagate from the clock definition point to a register clock pin. The rise and fall latencies are the latencies for rising and falling transitions at the register clock pin, respectively. Inversion of the clock waveform, if present in the clock network, is not taken into consideration when computing clock network latencies at register clock pins.

Clock source latency is the time it takes for a clock signal to propagate from its actual ideal waveform origin point to the clock definition point in the design. It can be used to model off-chip clock latency when the clock generation circuit is not part of the current design. You can use clock source latency for generated clocks to model the delay from master-clock to generated-clock definition point.

Dynamic clock latency is the component of the total clock latency which is considered 'dynamic' in nature. A 'dynamic' value may differ along successive clock cycles and hence may only be used to calculate CRP if a zero-cycle path is being considered. A zero-cycle path is one in which the same clock edge both launches and captures.

The dynamic component of any clock latency may be specified using the **-dynamic** switch. It may only

be specified if the **-source** switch and total clock latency is also specified with the command.

The phase error for output clocks coming out of phase locked loops (PLLs) can be specified using the **-pll_shift** switch. If **-pll_shift** option is used, the specified delay gets added to the phase adjustment of the PLL.

Constraint consistency assumes ideal clocking, which means clocks have a specified network latency (designated by the **set_clock_latency** command) or zero network latency, by default. Propagated clock network latency (designated by the **set_propagated_clock** command) is normally used for post-layout after final clock tree generation. Ideal clock network latency provides an estimate of the clock tree for pre-layout.

You can specify clock source latency for ideal or propagated clocks. The total clock latency at a register clock pin is the sum of clock source latency and clock network latency. If the rise and fall latencies are different, then the source latencies for a latch are picked based on the sense at the clock port and network latencies are picked based on the sense at the latch clock pin. Thus, source latency takes into account the clock inversions on the clock network, but network latency does not.

In bc_wc analysis mode, the -min and -max options specify the corner for the clock latency value (fast or slow), and the -early and -late options specify the early or late latency value within the corner. In the on_chip_analysis mode, the min/max operating conditions are the variation bounds within the single corner analysis. Therefore, only the min-early and max-late latencies are used for the analysis. In the on_chip_variation mode, it is sufficient to specify either -early/-late options or -min/-max options for set_clock_latency.

For internally-generated clocks, constraint consistency automatically computes the clock source latency provided that the master clock of the generated clock has propagated latency and no user-specified value for the generated clock source latency exists. If the master clock is ideal, the master clock has source latency, and there is no user-specified value for the generated clock's source latency, then zero source latency is assumed.

If the **set_clock_latency** command is applied to pins or ports, it affects all register clock pins in the transitive fanout of the pins or ports. Directly setting a network latency makes the affected registers ideal. A warning is issued if the **set_propagated_clock** command has previously set a propagated attribute on the same object (clock, pin, or port). Clock source latencies are allowed only on clocks and clock source pins.

No check will be performed at the UI level that a clock specified with **-clock** passes through the pin or pins referenced in object_list or not. If the clock specified does not pass through the pin, the command will have no effect but no warning of this fact is given.

To undo **set_clock_latency**, use the **remove_clock_latency** command.

To see clock network and source latency information (including dynamic component), use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example specifies a rise latency of **1.2** and a fall latency of **0.9** for a clock named **CLK1**.

```
ptc_shell> set_clock_latency 1.2 \
```

```
-rise [get_clocks CLK1]
ptc_shell> set_clock_latency 0.9 \
-fall [get_clocks CLK1]
```

The following example specifies an early rise and a fall source latency of **0.8** and a late rise and fall source latency of **0.9** for a clock named **CLK1**.

```
ptc_shell> set_clock_latency 0.8 -source \
-early [get_clocks CLK1]
ptc_shell> set_clock_latency 0.9 -source \
-late  [get_clocks CLK1]
```

The following example specifies an early and late source latency of 3 and 5 respectively with dynamic components of 0.5.

```
ptc_shell> set_clock_latency 3 -source \
-early -dynamic 0.5  [get_clocks CLK1]
ptc_shell> set_clock_latency 5 -source \
-late  -dynamic 0.5 [get_clocks CLK1]
```

SEE ALSO

remove_clock_latency(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
set_propagated_clock(2)

set_clock_map

Sets up custom clock mapping for block-to-top or SDC-to-SDC comparison.

SYNTAX

```
string set_clock_map
   -clocks1 clocks1_list
   -clocks2 clocks2_list
   [-scenario1 _first_scenario1]
   [-scenario2 s_second_cenario2]
   [-block_design block_design]
   [-cells cell_list]
```

Data Types

```
   clocks1_list   list
   clocks2_list   list
   cell_list      list
```

ARGUMENTS

-clocks1 ***clocks1_list***

A list of source clocks of the clock mapping in *scenario1*. This list must be the same length as the *clocks2_list*.

-clocks2 ***clocks2_list***

A list of destination clocks of the clock mapping in *scenario2*. This list must be the same length as the *clocks1_list*.

-scenario1 ***first_scenario1***

The first (source) scenario of the clock mapping. In block-to-top comparison, this must be the top scenario.

-scenario2 ***second_scenario2***

The second (destination) scenario of the clock mapping. In block-to-top comparison, this must be the block scenario.

-block_design *block_design*

The block design that the clock mapping applies to in block-to-top comparison. The clock mapping applies to all instances of the *design*. This option is ignored if the *-cells* option is specified.

-cells *cell_list*

A list of cell instances to apply the clock mapping to. This option is valid only for block to top clock mapping.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets up custom clock mapping for block-to-top or SDC-to-SDC comparison. For the clocks that are not mapped by this command, the default waveform based comparison is used to match the clocks.

EXAMPLES

The following example maps the top_Clk in top_scenario of top design to the blk_Clk of blk_scenario in block design, and then compares block constraints with top constraints using the custom clock map.

```
ptc_shell> current_design top
ptc_shell> set_clock_map -scenario1 top_scenario -scenario2 blk_scenario
-clocks1 {top_Clk} -clocks2 {blk_Clk} -block_design block
ptc_shell> compare_block_to_top -block_design block -use_clock_map
```

SEE ALSO

report_clock(2)
remove_clock_map(2)
compare_constraints(2)
compare_block_to_top(2)

set_clock_sense

The set_clock_sense command is deprecated. Use the set_sense command.

SEE ALSO

```
set_sense(2)
remove_sense(2)
```

set_clock_transition

Specifies transition time of register clock pins.

SYNTAX

```
string set_clock_transition [-rise]
   [-fall]
   [-min]
   [-max]
   transition
   clock_list

float transition
list clock_list
```

ARGUMENTS

-rise

Specifies clock transition time for rising clock edge.

-fall

Specifies clock transition time for falling clock edge.

-min

Specifies clock transition time for minimum conditions.

-max

Specifies clock transition time for maximum conditions.

transition

Specifies transition time of clock pins.

clock_list

Provides a list of clocks in the design. The transition times on all register clock pins in the transitive fanout of specified clock are affected. You only can specify clocks with ideal latency in the list. When register clock pins in the fanout of a clock are marked with propagated latency, **set_clock_transition**

values are ignored.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command provides the ability to override the calculated transition times on clock pins of registers.

This command is especially useful for pre-layout when clock trees are incomplete and calculated transition times at register clock pins can be highly pessimistic. The transition value specified with this command overrides the transition times of all nets directly feeding a sequential element clocked by the specified clock.

Use the **set_clock_transition** command only with ideal clocks. For propagated clocks the calculated transition times are used. Propagated clocks are only set after the final clock tree is constructed.

If a clock transition is not specified for an ideal clock, and if the variable timing_ideal_clock_zero_default_transition is TRUE (by default it is TRUE), then zero transition will be used. Else, the transition time is calculated as it is for other pins in the design.

To undo **set_clock_transition**, use **remove_clock_transition**.

To list all clock transition values which have been set, use **report_clock -skew**.

EXAMPLES

This example specifies a rise transition time of 0.38 and fall transition time of 0.25 for register clock pins clocked by "CLK1".

```
ptc_shell> set_clock_transition 0.38 -rise [get_clocks CLK1]
ptc_shell> set_clock_transition 0.25 -fall [get_clocks CLK1]
```

SEE ALSO

remove_clock_transition(2)
report_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)
set_propagated_clock(2)
timing_ideal_clock_zero_default_transition(3)

set_clock_uncertainty

Specifies the uncertainty (skew) of specified clock networks.

SYNTAX

```
string set_clock_uncertainty
   uncertainty
   [object_list |
    -from from_clock
      | -rise_from rise_from_clock
      | -fall_from fall_from_clock
    -to to_clock
      | -rise_to rise_to_clock
      | -fall_to fall_to_clock]
   [-rise]
   [-fall]
   [-setup]
   [-hold]

list from_clock
list rise_from_clock
list fall_from_clock
list rise_to_clock
list fall_to_clock
list to_clock
list object_list
float uncertainty
```

ARGUMENTS

-from *from_clock* -to *to_clock*

These two options specify the source and destination clocks for interclock uncertainty. You must specify either the pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to**, or *object_list*; you cannot specify both.

-rise_from *rise_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to rising edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-fall_from *fall_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to falling edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_to ***rise_to_clock***

Same as the **-to** option, but indicates that *uncertainty* applies only to rising edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-fall_to ***fall_to_clock***

Same as the **-to** option, but indicates that *uncertainty* applies only to falling edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

object_list

Specifies a list of clocks, ports, or pins for simple uncertainty; the uncertainty is applied either to capturing latches clocked by one of the clocks in *object list*, or capturing latches whose clock pins are in the fanout of a port or pin specified in *object_list*. You must specify either the pair of **-from** and **-to**, or *object_list*; you cannot specify both.

-rise

Indicates that *uncertainty* applies to only the rising edge of the destination clock. By default, the uncertainty applies to both rising and falling edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-rise_to** instead.

-fall

Indicates that *uncertainty* applies to only the falling edge of the destination clock. By default, the uncertainty applies to both rising and falling edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-fall_to** instead.

-setup

Indicates that *uncertainty* applies only to setup checks. By default, *uncertainty* applies to both setup and hold checks.

-hold

Indicates that *uncertainty* applies only to hold checks. By default, *uncertainty* applies to both setup and hold checks.

uncertainty

A floating point number that specifies the uncertainty value. Typically, clock uncertainty should be positive. Negative uncertainty values are supported for constraining designs with complex clock relationships. Setting the uncertainty value to a negative number could lead to optimistic timing analysis and should be used with extreme care.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies the clock uncertainty (skew characteristics) of specified clock networks. This command can specify either interclock uncertainty or simple uncertainty. For interclock uncertainty, use the **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to** options to specify the source clock and the destination clock; all paths between these receive the uncertainty value. For simple uncertainty, use *object_list*; the uncertainty value is either to capturing latches clocked by one of the clocks in *object list*, or capturing latches whose clock pins are in the fanout of a port or pin specified in *object_list*.

Set the uncertainty to the worst skew expected to the endpoint or between the clock domains. You can increase the value to account for additional margin for setup and hold.

When you specify interclock uncertainty, ensure that you specify it for all possible interactions of clock domains. For example, if you specify paths from CLKA to CLKB and CLKB to CLKA you must specify the uncertainty for both even if the values are the same. For an example, see the EXAMPLES section.

Interclock uncertainty is more specific than simple uncertainty. If a command that specifies interclock uncertainty conflicts with a command that specifies simple uncertainty, the command that specifies interclock uncertainty takes precedence. For an example, see the EXAMPLES section.

If there is no applicable interclock uncertainty for a path, the value for simple uncertainty is used. For an example, see the EXAMPLES section.

To remove the uncertainties set by **set_clock_uncertainty**, use the **remove_clock_uncertainty** command.

To view clock uncertainty information, use the **report_clock -skew** command.

EXAMPLES

The following example specifies that all paths leading to registers or ports clocked by CLK have setup uncertainty of 0.65 and hold uncertainty of 0.45.

```
ptc_shell> set_clock_uncertainty -setup 0.65 [get_clocks CLK]
ptc_shell> set_clock_uncertainty -hold 0.45 [get_clocks CLK]
```

The following example specifies interclock uncertainties between PHI1 and PHI2 clock domains.

```
ptc_shell> set_clock_uncertainty 0.4 -from PHI1 -to PHI1
ptc_shell> set_clock_uncertainty 0.4 -from PHI2 -to PHI2
ptc_shell> set_clock_uncertainty 1.1 -from PHI1 -to PHI2
ptc_shell> set_clock_uncertainty 1.1 -from PHI2 -to PHI1
```

The following example specifies interclock uncertainties between PHI1 and PHI2 clock domains with specific edges.

```
ptc_shell> set_clock_uncertainty 0.4 -rise_from PHI1 -to PHI2
ptc_shell> set_clock_uncertainty 0.4 -fall_from PHI2 -rise_to PHI2
ptc_shell> set_clock_uncertainty 1.1 -from PHI1 -fall_to PHI2
```

The following example shows conflicting **set_clock_uncertainty** commands, one for simple uncertainty and one for interclock uncertainty. The interclock uncertainty value of 2 takes precedence.

```
ptc_shell> set_clock_uncertainty 5 [get_clocks CLKA]
```

```
ptc_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] -to [get_clocks CLKA]
```

The following example specifies the uncertainty from CLKA to CLKB and from CLKB to CLKA. Notice that both must be specified even though the value is the same for both.

```
ptc_shell> set_clock_uncertainty 2 -from [get_clocks CLKA] -to [get_clocks CLKB]
ptc_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] -to [get_clocks CLKA]
```

The following example illustrates a situation in which simple uncertainty is used when there is no applicable interclock uncertainty for a path. The first command specifies a simple uncertainty of 5 for CLKA paths, and the second command specifies an interclock uncertainty of 2 for paths from CLKB to CLKA. If there are paths between CLKA and other clocks (for example, CLKC, CLKD, ...) for which interclock uncertainty has not been specifically defined, the simple uncertainty (in this case, 5) is used.

```
ptc_shell> set_clock_uncertainty 5 [get_clocks CLKA]
ptc_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] -to [get_clocks CLKA]
```

SEE ALSO

remove_clock_uncertainty(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)

set_current_command_mode

SYNTAX

```
string set_current_command_mode

   -mode command_mode | -command command

string command_mode
string command
```

ARGUMENTS

-mode ***command_mode***

Specifies the name of the new command mode to be made current. **-mode** and **-command** are mutually exclusive; you must specify one, but not both.

-command ***command***

Specifies the name of the command whose associated command mode is to be made current. **-mode** and **-command** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

The **set_current_command_mode** sets the current command mode. If there is a current mode in effect, the current mode is first canceled, causing the associated clean up to be executed. The initialization for the new command mode is then executed as it is made current.

If the name of the given command mode is empty, the current command mode is cancelled and no new command mode is made current.

A current command mode stays in effect until it is displaced by a new command mode or until it is cleared by an empty command mode.

If the command completes successfully, the new current command mode name is returned. On failure,

the command returns the previous command mode name which will remain current and prints an error message unless error messages are suppressed.

EXAMPLES

The following example sets the current command mode to a mode called "mode_1"

```
shell> set_current_command_mode -mode mode_1
```

The following example clears the current command mode without setting a new mode.

```
shell> set_current_command_mode -mode ""
```

The following example sets the current command mode to the mode associated with the command "modal_command"

```
shell> set_current_command_mode -command modal_command
```

set_data_check

Sets data-to-data checks using the specified values of setup and hold time.

SYNTAX

```
string set_data_check
   {-from from_object
      | -rise_from from_object
      | -fall_from from_object}
   {-to to_object
      | -rise_to to_object
      | -fall_to to_object}
   [-setup | -hold]
   [-clock clock_object]
   [check_value]
```

Data Types

from_object	object
to_object	object
clock_object	object
check_value	float

ARGUMENTS

-from *from_object*

Specifies a pin or port in the current design as the related pin of the data-to-data check to be set. Both rising and falling delays are checked. You must specify either the *-from*, *-rise_from*, or *-fall_from* option.

-rise_from *from_object*

Similar to the *-from* option, but applies only to rising delays at the related pin. You must specify either the *-from*, *-rise_from*, or *-fall_from* option.

-fall_from *from_object*

Similar to the *-from* option, but applies only to falling delays at the related pin. You must specify either the *-from*, *-rise_from*, or *-fall_from* option.

-to *to_object*

Specifies a pin or port in the current design as the constrained pin of the data-to-data check to be set. Both rising and falling delays are constrained. You must specify either the *-to*, *-rise_to*, or *-fall_to* option.

-rise_to *to_object*

Similar to the *-to* option, but applies only to rising delays at the constrained pin. You must specify either the *-to*, *-rise_to*, or *-fall_to* option.

-fall_to *to_object*

Similar to the *-to* option, but applies only to falling delays at the constrained pin. You must specify either the *-to*, *-rise_to*, or *-fall_to* option.

-setup

Indicates that the data check value is for setup analysis only. If neither the *-setup* nor *-hold* option is specified, the value applies to both setup and hold.

-hold

Indicates that the data check value is for hold analysis only. If neither the *-setup* nor *-hold* option is specified, the value applies to both setup and hold.

-clock *clock_object*

Specifies the name of a single clock to be used for the data check. Use this option only if your design has multiple clocks per register and you want to select a single clock to be used. For more details about multiple clocks per register, see the DESCRIPTION section.

check_value

Specifies the value of the setup and hold time for the check.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_data_check** command specifies a data-to-data check to be performed between the from object and to object using the specified setup and hold value.

The data check is treated as nonsequential; that is, the path goes through the related and constrained pins and is not broken at these pins. To remove information set by the **set_data_check** command, use the **remove_data_check** command.

EXAMPLES

The following example creates a data-to-data check from and1/B to and1/A with respect to the rising edge of signal at and1/B, using a setup and hold time of 0.4.

```
ptc_shell> set_data_check -rise_from and1/B -to and1/A 0.4
```

The following example creates a data-to-data check from and1/B to and1/A with respect to the rising edge of signal at and1/B, coming from the clock domain of CK1, constraining only the falling edge of signal at and1/A, using a setup time of 0.5 and no hold check.

```
ptc_shell> set_data_check -rise_from and1/B -fall_to and1/A -setup \
          -clock [get_clock CK1] 0.5
```

SEE ALSO

remove_data_check(2)

set_disable_clock_gating_check

Disables the clock gating check for specified objects in the current design.

SYNTAX

```
string set_disable_clock_gating_check  object_list

list object_list
```

ARGUMENTS

object_list

Specifies a list of cells and pins for which the clock gating check is to be disabled.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Disables the clock gating check for specified cells and pins in the current design. This command will only disable auto-inferred clock gating checks. Clock gating checks from library will not be disabled.

Any cell or pin in the current design or its subdesigns can be disabled. Cells at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name*. Pins at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name/pin_name*.

Note: For subdesigns in the hierarchy, you must specify instance names instead of design names.

When the checking through a cell is disabled, all gating checks in the cell are disabled. When the checking through a pin is disabled, any gating check is disabled if it uses the disabled pin as a gating clock pin or a gating enable pin.

To restore gating checks disabled by **set_disable_clock_gating_check**, use **remove_disable_clock_gating_check**. To globally disable all clock gating checks, set the **timing_disable_clock_gating_checks** variable to *true*.

EXAMPLES

The following example disable gating checks on the specified cell or pin.

ptc_shell> **set_disable_clock_gating_check an2/Z**

ptc_shell> **set_disable_clock_gating_check or1**

SEE ALSO

remove_disable_clock_gating_check(2)
timing_disable_clock_gating_checks(3)

set_disable_timing

Disables timing arcs in a circuit.

SYNTAX

```
string set_disable_timing
   [-from from_pin_name -to to_pin_name]
   object_list

string from_pin_name
string to_pin_name
list object_list
```

ARGUMENTS

-from "*from_pin_name*"

Specifies that arcs only from this pin on the specified cell are to be disabled. The *from_pin_name* value must be a valid library pin name corresponding to the cell in *object_list*. When used, the **-from** and **-to** options must be specified together. The *object_list* must contain only cells, and the command specifies that arcs only between these two pins on the specified cells are to be disabled.

-to "*to_pin_name*"

Specifies that arcs only to this pin on the specified cell are to be disabled. The *to_pin_name* value must be a valid library pin name corresponding to the cell in *object_list*. When used, the **-from** and **-to** options must be specified together. The *object_list* must contain only cells, and the command specifies that arcs only between these two pins on the specified cells are to be disabled.

object_list

Specifies a list of cells, pins, lib-cells, lib-pins, ports, or timing-arcs to be disabled. This list can include library objects.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Disables timing through the specified cells, pins, ports, or timing arcs in the **current_design**.

Any cell, pin, port, or timing arc in the **current_design** or its subdesigns can be disabled. Cells at lower levels in the design hierarchy are specified as *instance1*/*instance2*/*cell_name*. Pins at lower levels in the design hierarchy are specified as *instance1*/*instance2*/*cell_name*/*pin_name*. Ports at lower levels in the design hierarchy are specified as *instance1*/*instance2*/*cell_name*/*port_name*. Timing arcs have to be collected using the *get_timing_arcs* command.

Note: For subdesigns in the hierarchy, you must specify instance names instead of design names.

When the timing through a cell is disabled, only those cell arcs are disabled that lead to the output pins of the cell. When the timing through a pin or port is disabled, only those cell arcs that lead from a load pin and to a driver pin are disabled. For bidirectional pins or ports the cell arcs from the load side and to the driver side are disabled.

When **-from** and **-to** pins are specified, all arcs between these two pins on the cell are disabled.

To view disabled timing arcs in the current design, use the **report_disable_timing** command. To restore timing arcs disabled by **set_disable_timing**, use the **remove_disable_timing** command. The **reset_design** command removes all user-specified attributes from the current design, including those set by **set_disable_timing**.

EXAMPLES

```
ptc_shell> set_disable_timing  -from A  -to Z  [ get_cells { CO } ]
ptc_shell> set_disable_timing [get_timing_arcs -from { CO/A } -to { CO/Z } ]
```

SEE ALSO

remove_disable_timing(2)
report_timing(2)
reset_design(2)
get_timing_arcs(2)

set_dont_touch

Specifies that the given cells, nets, designs, and library cells should not be optimized.

SYNTAX

```
string set_dont_touch
   object_list
   [value]
```

Data Types

```
   object_list   list
   value         Boolean
```

ARGUMENTS

object_list

Specifies a list of cells, nets, designs, or library cells on which to place the **dont_touch** attribute.

value

Specifies the value with which to set the **dont_touch** attribute. Allowed values are **true** (the default) or **false**.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the **dont_touch** attribute on cells and nets in the current design, or on designs and library cells.

For a complete description of the **dont_touch** attribute and its effect, see the man page of the **set_dont_touch** command in Design Compiler.

EXAMPLES

The following commands specify **set_dont_touch** on the block1 and analog1 cells, but not on net N1.

```
ptc_shell> set_dont_touch [get_cells {block1 analog1}]
1
ptc_shell> set_dont_touch [get_nets N1] false
1
```

SEE ALSO

set_dont_touch_network(2)

set_dont_touch_network

Specifies that the given clock networks or pins/ports should not be optimized.

SYNTAX

```
string set_dont_touch_network
   [-no_propagate]
   [-clear]
   object_list
```

Data Types

```
   object_list   list
```

ARGUMENTS

-no_propagate

Specifies that the **dont_touch_network** attribute should not propagate through logic gates.

-clear

Specifies that the **dont_touch_network** attribute should be removed.

object_list

Specifies a list of clocks, pins, or ports.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the **dont_touch_network** attribute on clocks, pins, or ports in the current design.

The **set_dont_touch_network** command is intended primarily for clock circuitry.

See the man page of the **set_dont_touch** command in Design Compiler for a complete description of the **dont_touch** attribute and its effect.

EXAMPLES

The following command adds a **dont_touch_network** attribute to the port named clock_in.

```
ptc_shell> set_dont_touch_network [get_ports clock_in]
```

SEE ALSO

set_dont_touch(2)

set_drive

Sets the resistance to a specified value on specified input or inout ports in the current design.

SYNTAX

```
string set_drive
   [-rise] [-fall]
   [-min] [-max]
   resistance_value port_list

float resistance_value
list port_list
```

ARGUMENTS

-rise

Indicates that *resistance_value* is to be used to drive the ports only for the rising case.

-fall

Indicates that *resistance_value* is to be used to drive the ports only for the falling case.

-min

Applies only to designs in min-max mode (min and max operating conditions). Indicates that the *resistance_value* is the minimum resistance.

-max

Applies only to designs in min-max mode (min and max operating conditions). Indicates that the *resistance_value* is the maximum resistance.

resistance_value

Specifies a nonnegative port drive resistance value for the ports in *port_list*. The value must be >= 0; units must be the same as those in the technology library.

port_list

Specifies a list of input or inout ports in the current design, on which the *resistance_value* is to be set.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the resistance to a specified value on specified input or inout ports in the current design. Constraint consistency models the driver as a resistance and uses that value to calculate the wire delay of the port. To view the drive that is set, use **report_port -drive**.

Depending on the options used, **set_drive** sets these attributes on the specified ports:

drive_resistance_fall_max
drive_resistance_fall_min
drive_resistance_rise_max
drive_resistance_rise_min

If **set_drive** is issued without any options, the **drive_resistance_fall_max** and **drive_resistance_rise_max** attributes are set; in min-max mode, the other two attributes are also set.

If **set_drive** is issued with only the **-rise** option, only the **drive_resistance_rise_max** attribute is set; in min-max mode, **drive_resistance_rise_min** is also set.

If **set_drive** is issued with only the **-fall** option, only the **drive_resistance_fall_max** attribute is set; in min-max mode, **drive_resistance_fall_min** is also set.

Drive resistance is the output resistance of the cell that drives the port, so that a higher drive resistance means less drive capability and longer delays. Thus, a drive *resistance_value* of 0 is infinite drive, or no delay between the ports and all ports connected to them. Setting *resistance_value* to 0 is the same as removing the drive resistance with **remove_drive_resistance**.

EXAMPLES

The following example sets the drive resistance to 5 on all input ports in the current design, and displays the ports on which the drive resistance has been set.

```
ptc_shell> set_drive 5 [all_inputs]
1
ptc_shell> report_port -drive
****************************************
Report : port
        -drive
Design : counter
****************************************

                Resistance          Transition
Input Port    Rise      Fall      Rise      Fall
-------------------------------------------------------------
A             5.00      5.00      --        --
B             5.00      5.00      --        --
C             5.00      5.00      --        --
CL            5.00      5.00      --        --
```

```
CLK             5.00      5.00      --        --
D               5.00      5.00      --        --
JOKE            5.00      5.00      --        --
L               5.00      5.00      --        --
P               5.00      5.00      --        --
RESET           5.00      5.00      --        --
T               5.00      5.00      --        --
```

SEE ALSO

remove_drive_resistance(2)
report_port(2)
set_load(2)

set_driving_cell

Sets the port driving cell.

SYNTAX

```
string set_driving_cell
   [-lib_cell lib_cell_name]
   [-rise]
   [-fall]
   [-min]
   [-max]
   [-library lib_name]
   [-pin pin_name]
   [-from_pin from_pin_name]
   [-multiply_by factor]
   [-dont_scale]
   [-no_design_rule]
   [-input_transition_rise rtrans]
   [-input_transition_fall ftrans]
   [-clock clock_name]
   [-clock_fall]
   port_list

string lib_cell_name
string lib_name
string pin_name
string pin_name
float factor
float rtran
float ftran
[-clock clock_name]
[-clock_fall]
list port_list
```

ARGUMENTS

-lib_cell *lib_cell_name*

Specifies the name of the library cell used to drive the ports. You must use the **-pin** option if the cell has more than one output pin. If different cells are needed for the rising and the falling cases, use separate commands with the **-rise** or **-fall** option. Use the **-from_pin** option to choose between multiple input pins with arcs to this output pin. This option is required.

-rise

Sets driving_cell information for only the rising port transition.

-fall

Sets driving_cell information for only the falling port transition.

-min

Sets driving_cell information for only the minimum analysis. This option is valid only in min-max mode.

-max

Sets driving_cell information for only the maximum analysis. This option is valid even when not in min-max mode. When not in min-max mode, the option is not required because it is the default.

-library ***lib_name***

Specifies the name of the library containing the *library_cell_name* value. This is the library of the driving cell. By default, the libraries in **link_library** are searched for the cell.

-pin ***pin_name***

Specifies the output pin whose drive is used. This is the driving pin name. If you do not include the **-from_pin** option, the command uses an arbitrary pin arc ending at the specified pin.

-from_pin ***from_pin_name***

Specifies an input pin on the specified cell so the command uses the drive of the timing arc from this pin to the specified pin.

-multiply_by ***factor***

Multiplies the calculated transition time by the specified multiplier. The valid transition multiplier range is greater than or equal to 0.

-dont_scale

Prevents operating condition scaling. Indicates that the timing analyzer is not to scale the drive capability of the ports according to the current operating conditions. By default, the port drive capability is scaled for operating conditions exactly as the driving cell itself would have been scaled.

-no_design_rule

Specifies not to transfer design rules from the driving cell to the port. If you do not include this option, the design rules (such as max_capacitance) of the library pin are applied to the port.

-input_transition_rise ***rtran***

Specifies the input rising transition time associated with the **-from_pin** option. If you do not include this option, the default value is 0. Use the **-input_transition_rise** and **-input_transition_fall** options to capture the accurate transition time associated with the *from_pin_name* value. This can obtain more accurate information on the transition time and delay time at the output pin.

-input_transition_fall ***ftran***

Specifies the input falling transition time associated with the *from_pin_name* value. If you do not include this option, the default value is 0.

-clock *clock_name*

The driving cell is set relative the specified clock.

-clock_fall

Specifies that driving cell is relative to the falling edge of the clock. The default is the rising edge.

port_list

Provides a list of input ports. The list contains input or inout port names in the current design on which the driving cell information is set.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the port driving cell. Sets attributes on the specified input or inout ports in the current design to associate an external driving cell with the ports. The drive capability of the port is the same as if the specified driving cell were connected in the same context to allow accurate modeling of port drive capability for nonlinear delay models. If you do not include the **-dont_scale** option, the drive capability of the port is scaled according to the current operating conditions. Use the **report_port** command with the **-drive** option to view drive information on ports.

The driving cell can be relative to a clock by using -clock option. This means the driving cell apply only to those external paths driven by the clock. Using -clock or -clock_fall option can be meaningful only when timing_slew_propagation_mode is in worst_arrival mode. Note if it is in worst_slew mode, there must be a driving cell without any clock specified to see any driving cell on the port. If a clock-relative driving cell is set on a port, these driving-cell attributes will not be accessible via **get_attribute** until -clock and -clock_fall options are supported for set_driving_cell in SDC.

Use the **remove_driving_cell** command to remove driving cell information from ports.

Note: The **set_driving_cell** command removes any corresponding drive resistance or input transition attributes (from the **set_drive_resistance** command or the **set_input_transition** command) on the specified ports. If possible, always use the **set_driving_cell** command instead of the **set_drive_resistance** command, because **set_driving_cell** allows accurate calculation of port delay and transition time for library cells with nonlinear dependence on capacitance.

EXAMPLES

```
ptc_shell> set_driving_cell \
-lib_cell AND [get_ports A]
ptc_shell> report_port -drive A

****************************************
Report : port -drive
Design : counter
Version: ...
Date : Tue 1996
****************************************
```

Resistance Transition Input Port Rise Fall Rise Fall ---------- A -- -- -- --

Driving Cell Input Port Rise Fall Mult Attrs ---------- ------- A AND AND --

SEE ALSO

all_inputs(2)
remove_driving_cell(2)
report_port(2)
reset_design(2)
set_capacitance(2)
set_drive_resistance(2)
set_input_transition(2)

set_false_path

Identifies paths in a design that are to be marked as false, so that they are not considered during timing analysis.

SYNTAX

```
Boolean set_false_path
   [-setup] [-hold]
   [-rise] [-fall]
   [-reset_path]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-comment comment_string]

list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
string  comment_string
```

ARGUMENTS

-setup

Indicates that setup (maximum) paths are to be marked as false. This option disables setup checking for specified paths. If you do not specify either **-setup** or **-hold**, both setup and hold timing are marked false.

-hold

Indicates that hold (minimum) paths are to be marked as false. This option disables hold checking for

specified paths. If you do not specify either **-setup** or **-hold**, both setup and hold timing are marked false.

-rise

Indicates that rising delays are to be marked as false, as measured on the path endpoint. If you do not specify either **-rise** or **-fall**, both rise and fall timing are marked false.

-fall

Indicates that falling delays are to be marked as false, as measured on the path endpoint. If you do not specify either **-rise** or **-fall**, rise and fall timing are marked false.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise/fall setup/hold type is reset. This is equivalent to using the **reset_path** command with similar arguments before the **set_false_path** command is issued.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified.If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-through ***through_list***

Specifies a list of pins, ports, cells or nets through which the disabled paths must pass. You can specify **-through** more than once in one command invocation. Nets are interpreted to imply the net segment's local driver pins.

If you do not specify any type of **through** option, all timing paths specified using the **-from** and **-to** options are affected.

For a detailed discussion of the use of multiple **-through**, **-rise_through**, and **-fall_through** options, see the DESCRIPTION section.

-rise_through ***rise_through_list***

The same as the **-through** option, but the paths must have a rising transition at the through points.

If you do not specify any type of **-through** option, all timing paths specified using the **-from** and **-to** options are affected.

For a detailed discussion of the use of multiple **-through**, **-rise_through**, and **-fall_through** options, see the DESCRIPTION section.

-fall_through ***fall_through_list***

The same as the **-through** option, but the paths must have a falling transition at the through points.

If you do not specify any type of **-through** option, all timing paths specified using the **-from** and **-to** options are affected.

For a detailed discussion of the use of multiple **-through**, **-rise_through**, and **-fall_through** options, see the DESCRIPTION section.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If you specify a cell, one path endpoint on that cell is affected.

You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_false_path** command marks startpoint/endpoint pairs as false timing paths; that is, paths that cannot propagate a signal. The command removes timing constraints on these false paths, so that they

are not considered during timing analysis. Path startpoints are input ports or register clock pins; path endpoints are register data pins or output ports. The command disables maximum delay (setup) checking and minimum delay (hold) checking for the specified paths.

The **set_false_path** command is a point-to-point timing exception command, and overrides the default single-cycle timing relationship for one or more timing paths. False path information always takes precedence over multicycle path information. Also, **set_false_path** commands override **set_max_delay** and **set_min_delay** commands.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

In some earlier releases of constraint consistency, a single **-through** option specifies multiple traversal points. You can revert to this behavior by setting the **timing_through_compatability** variable to true. If you do so, the behavior is as illustrated in the following example, which specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through {B1 C1} -to D1
```

In this mode, you cannot use more than one **-through** option in a single command.

To disable the timing at a particular cell along a path, use the **set_disable_timing** command.

To remove the false path designations set by **set_false_path**, use the **reset_path** command.

For crosstalk analysis the false paths are treated as sensitizable. So, the aggressors on the false paths could still effect its victims. When the clocks are asynchronous to each other, **set_clock_groups -async** command should be used instead of **set_false_paths** between the async clocks. With out **set_clock_groups -async** command, the clocks will be treated to be sync to each other.

EXAMPLES

The following example disables timing paths from ff12 to ff34.

ptc_shell> **set_false_path -from ff12 -to ff34**

The following example disables rising timing paths from U14/Z to ff29/Reset.

ptc_shell> **set_false_path -rise_from U14/Z -to ff29/Reset**

The following examples disables hold checking for endpoints clocked by CLK1.

ptc_shell> **set_false_path -hold -to [get_clocks CLK1]**

The following example disables all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C}.

ptc_shell> **set_false_path -from ff1/CP -through {U1/Z U2/Z}**
-through {U3/Z U4/C} -to ff2/D

The following example disables all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall to one or more of {U3/Z U4/C}.

ptc_shell> **set_false_path -from ff1/CP**
-rise_through {U1/Z U2/Z}
-fall_through {U3/Z U4/C} -to ff2/D

SEE ALSO

current_design(2)
reset_design(2)
reset_path(2)
set_clock_groups(2)
set_disable_timing(2)
set_max_delay(2)
set_min_delay(2)
set_multicycle_path(2)

set_fanout_load

Sets fanout_load for output ports in the current design.

SYNTAX

```
string set_fanout_load
   value
   port_list
```

Data Types

```
   value            float
   port_list        list
```

ARGUMENTS

value

Shows the **fanout_load** value, in units consistent with the **fanout_load** and **max_fanout** values in the technology library.

port_list

Provides a list of output ports.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies a **fanout_load** for output ports in the current design. By default, ports are considered to have fanout_load of 0.0. Fanout load for a net is the sum of **fanout_load** attributes for all input pins and output ports connected to the net. Output pins may have maximum fanout limits, specified in the library or with the **set_max_fanout** command.

To remove **fanout_load** information from ports, use the **remove_fanout_load** command.

EXAMPLES

This example sets the fanout_load on ports matching "OUT*" to 3.0.

```
ptc_shell> set_fanout_load 3.0 "OUT*"
```

SEE ALSO

```
remove_fanout_load(2)
set_max_fanout(2)
```

set_hierarchical_config

Specifies the configuration for constraint consistency to perform the hierarchical constraint analysis flow.

SYNTAX

```
string set_hierarchical_config
   -block name_of_sub_block
   [-instances names_of_instances]
```

Data Types

```
   name_of_sub_block   string
   names_of_instances  list
```

ARGUMENTS

-block *name_of_sub_block*

Specifies the name of the design sub-block which needs to be considered for the analyze_design command. Only these sub-blocks are of interest to the analyze_design rules which needs the block information.

-instances *names_of_sub_block_instances*

Specifies a list of one or more instance names in the current design which are to be considered for the analyze_design rules.

These named instances must have the same reference block which is specified by the option **-block** in the same configuration command.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Some of the analyze_design rules which are part of the hierarchical ruleset need the specific information about the sub-blocks which are of interest to the current analyze_design run. These blocks and instances of those blocks needs to be specified by the user.

This command is used to name the blocks and the instances of those blocks which are needed for the analyze_design rules.

EXAMPLES

The following example assumes a design with 2 levels of hierarchy which the highest level is 'TOP' and the lowest level of 'BOT'. The sub-block 'BOT' has two instances 'b1' and 'b2'.

To specify the block and the instances which are to be considered for the analyze_design command.

```
ptc_shell> set_hierarchical_config  -block BOT -instances [list b1 b2]
```

SEE ALSO

link_design(2)
analyze_design(2)
report_constraint_analysis(2)
save_session(2)

set_host_options

Specifies host options for compute resources.

SYNTAX

```
int set_host_options
   -max_cores number
   [-submit_command command]
   [-num_processes number]
   [host_names]
```

Data Types

```
   number         integer
```

ARGUMENTS

-max_cores *number*

Specifies the CPU cores usage limit for the current process. The *number* value is currently limited to the range between 1 and 32.

-submit_command *command*

Specifies the command to call when submitting and launching the constraint consistency processes associated with the host options. This option specifies the path and command to call to perform this operation. When it is called, the job and process ID of the job is captured and associated with the host options. If the *-submit_command* and *host_name* options are not specified, or the *host_name* is the same as where the master process is running or is localhost, the processes are launched locally by the operating system of the master.

-num_processes *number*

Specifies the number of processes to launch.

host_names

Specifies a list of names of hosts on which to launch the slave processes. The default name for the host on which the current process is running is called localhost. If host names are specified and if any of the names are not localhost, or the name of the host on which the master process is running, the -

submit_command option can be used to specify the command to setup a remote connection to the hosts to launch the constraint consistency slave processes.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command is used to specify the host options of the current process. When constraint consistency is started, the initial limit is set as follows:

The core usage limit is set by using the minimum of the limit for a single PrimeTime SI license and the physical number of cores on the machine. The default single license is equivalent to a core limit of 4. For example, if you launch a constraint consistency shell on a 2-core machine, the initial core limit is 2. The core usage limit would be 4 if the machine had 8 cores.

When the **set_host_options** command is issued during a session, the host core usage limit is set to the minimum of the physical number of cores on the machine, the number of additional PrimeTime SI licenses that can be obtained, and the user-defined number.

EXAMPLES

The following example sets the maximum cores that can be used by constraint consistency to be 8.

```
ptc_shell> set_host_options -max_cores 8
Information: Checked out license 'GalaxyConstraint' (1) (GCA-001)
Information: The max_cores limit for the local (current) process has been
modified by 4 to 8. (CMCR-001)
1
```

SEE ALSO

list_licenses(2)

set_ideal_latency

Specifies ideal latency values for the pins in an ideal network.

SYNTAX

```
int set_ideal_latency
   [-rise] [-fall]
   [-min] [-max]
   value
   object_list
```

Data Types

```
   value          float
   object_list    list
```

ARGUMENTS

-rise

Indicates that the *value* option represents the rise latency time. If you do not specify the *-rise* or *-fall* options, both values are set.

-fall

Indicates that the *value* option represents the fall latency time. If you do not specify the *-rise* or *-fall* option, both values are set.

-min

Indicates that the *value* option represents the minimum latency time. If you do not specify the *-min* or *-max* option, both values are set.

-max

Indicates that the *value* option represents the maximum latency time. If you do not specify the *-min* or *-max* option, both values are set.

value

Specifies an ideal latency value on leaf cell pins or top-level ports in an ideal network.

object_list

Specifies a list of leaf cell pins and top-level ports on which ideal latency is set.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets an ideal latency value on leaf cell pins and top-level ports of an ideal network.

Ideal networks are normally used during pre-layout to avoid unnecessary DRCs. The specified ideal latency value provides an estimate of the latency time in the ideal network for pre-layout.

You can use the **set_ideal_latency** command for pins at lower levels of the design hierarchy.

To remove the ideal latency values from a design, use the **remove_ideal_latency** command.

EXAMPLES

The following example specifies a rise latency of *1.2* and a fall latency of *0.9* for ports named *A*, *B*, and *C*.

```
ptc_shell> set_ideal_latency 1.2 -rise {A B C}
ptc_shell> set_ideal_latency 0.9 -fall {A B C}
```

SEE ALSO

```
remove_ideal_latency(2)
remove_ideal_network(2)
remove_ideal_transition(2)
set_ideal_transition(2)
set_ideal_network(2)
```

set_ideal_network

Marks a set of ports or pins in the design as sources of an ideal network. This disables timing update of cells and nets in the transitive fanout of the specified objects.

SYNTAX

```
int set_ideal_network
   [-no_propagate]
   object_list
```

Data Types

object_list	list

ARGUMENTS

-no_propagate

Indicates that the ideal network is not propagated through leaf cells. Ideal properties are enabled on all nets that are electrically connected to the ideal network sources.

object_list

Specifies a list of objects to mark as the sources of an ideal network. Ideal network source objects may be ports or pins of leaf cells at any hierarchical level of the design. If nets are specified in the *object_list*, all of the nets' global driver pins are marked as ideal network sources. The **set_ideal_network** command accepts nets only when you specify the *-no_propagate* option.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command marks a set of ports or pins in the design as sources of an ideal network. You specify only the source of the network. Constraint consistency marks all pins in the transitive fanout of ideal sources as ideal. The ideal property is automatically spread by the tool and spread again, as necessary, after netlist

changes. The criteria for propagating the ideal property, starting at the source pins and ports, are as follows:

- A pin is marked as ideal if you specify it by using the **set_ideal_network** command, if it is either a driver pin there is a timing arc coming from an ideal pin or if it is a load pin that is driven by an ideal pin.
- Propagation traverses through combinational cells but stops at sequential cells.
- Ideal networks can be specified in the clock or data network.

The latency and transition times of an ideal network are zero by default, but you can override them by using the **set_ideal_latency** and **set_ideal_transition** commands.

To reverse the effect of the **set_ideal_network** command, use the **remove_ideal_network** command and specify the source pins or ports for the network. All ideal networks are removed using the **reset_design** command.

EXAMPLES

The following example creates an ideal network on ports in a design called 'TOP'.

```
ptc_shell> current_design {TOP}
ptc_shell> set_ideal_network {IN1 IN2}
```

SEE ALSO

remove_ideal_network(2)
reset_design(2)
set_ideal_latency(2)
set_ideal_transition(2)

set_ideal_transition

Specifies ideal transition values for the pins in an ideal network.

SYNTAX

```
int set_ideal_transition
   [-rise] [-fall]
   [-min] [-max]
   value
   object_list
```

Data Types

value	float
object_list	list

ARGUMENTS

-rise

Indicates that the *value* option represents the rise transition time. If you do not specify the *-rise* or *-fall* option, both values are set.

-fall

Indicates that the *value* option represents the fall transition time. If you do not specify the *-rise* or *-fall*option, both values are set.

-min

Indicates that the *value* option represents the minimum transition time. If you do not specify the *-min* or *-max*, both values are set.

-max

Indicates that the *value* option represents the maximum transition time. If you do not specify the *-min* or *-max* option, both values are set.

value

Specifies an ideal transition value on leaf cell pins or top-level ports in an ideal network.

object_list

Specifies a list of leaf cell pins and top-level ports on which ideal transition is set.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets an ideal transition value on leaf cell pins and top-level ports of an ideal network.

You can use the **set_ideal_transition** command for pins at lower levels of the design hierarchy.

To remove the ideal transition values from a design, use the **remove_ideal_transition** or **reset_design** command.

EXAMPLES

The following example specifies a rise transition of *1.2* and a fall transition of *0.9* for ports named *A*, *B*, and *C*.

```
ptc_shell> set_ideal_transition 1.2 -rise {A B C}
ptc_shell> set_ideal_transition 0.9 -fall {A B C}
```

SEE ALSO

```
remove_ideal_latency(2)
remove_ideal_network(2)
remove_ideal_transition(2)
set_ideal_latency(2)
set_ideal_network(2)
```

set_input_delay

Defines the arrival time relative to a clock.

SYNTAX

```
string set_input_delay
   [-clock clock_name]
   [-reference_pin pin_port_name]
   [-clock_fall]
   [-level_sensitive]
   [-rise]
   [-fall]
   [-max]
   [-min]
   [-add_delay]
   [-network_latency_included]
   [-source_latency_included]
   delay_value
   port_pin_list
```

Data Types

```
   clock_name              list
   pin_port_name           list
   delay_value             float
   port_pin_list           list
```

ARGUMENTS

-clock *clock_name*

Specifies the clock to which the specified delay is related. If you use the *-clock_fall* option, you must specify the *-clock clock_name* option. If you do not specify the *-clock* option, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with a period determined by considering the sequential cells in the transitive fanout of each port.

-reference_pin *pin_port_name*

Specifies the clock pin or port to which the specified delay is related. If you use this option and the propagated clocking is being used, the delay value is related to the arrival time at the specified reference pin, which is clock source latency plus its network latency from the clock source to this reference pin. The *-network_latency_included* and *-source_latency_included* options cannot be used at

the same time as the *-reference_pin* option. For ideal clock network, only source latency is applied.

The pin specified with the *-reference_pin* option should be a leaf pin or port in a clock network, in the direct or transitive fanout of a clock source specified with the *-clock* option. If multiple clocks reach the port or pin where you are setting the input delay and if the *-clock* option is not used, the command considers all of the clocks.

-clock_fall

Specifies that the delay is relative to the falling edge of the clock. When it is used with *-reference_pin* option, the delay is relative to the falling transition of the reference pin. The default is the rising edge or rising transition of a reference pin.

-level_sensitive

Specifies that the source of the delay is a level-sensitive latch. This allows the tool to derive setup and hold relationship for paths from this port as if it were a level-sensitive latch. If you do not use the *-level_sensitive* option, the input delay is treated as if it were a path from a flip-flop.

-rise

Specifies that *delay_value* refers to a rising transition on specified ports of the current design. If you do not specify the *-rise* or *-fall* option, rising and falling delays are assumed to be equal.

-fall

Specifies that *delay_value* refers to a falling transition on specified ports of the current design. If you do not specify the *-rise* or *-fall* option, rising and falling delays are assumed to be equal.

-max

Specifies that *delay_value* refers to the longest path. If you do not specify the *-max* or *-min* option, maximum and minimum input delays are assumed to be equal.

-min

Specifies that *delay_value* refers to the shortest path. If you do not specify the *-max* or *-min* option, maximum and minimum input delays are assumed to be equal.

-add_delay

Specifies whether to add delay information to the existing input delay or to overwrite. Use the *-add_delay* option to capture information about multiple paths leading to an input port that are relative to different clocks or clock edges.

For example, **set_input_delay 5.0 -max -rise -clock phi1 {A}** removes all other maximum rise input delay from "A", because the *-add_delay* option is not specified. Other input delays with different clocks or with clock_fall are removed.

In another example, the *-add_delay* option is specified as **set_input_delay 5.0 -max -rise -clock phi1 -add_delay {A}**. If there is an input maximum rise delay for "A" relative to clock "phi1" rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is maximum rise input delay relative to a different clock or different edge of the same clock, it remains with the new delay.

-network_latency_included

Specifies whether the clock network latency should not be added to the input delay value. If this option

is not specified, the clock network latency of the related clock is added to the input delay value. It has no effect if the clock is propagated or the input delay is not specified with respect to any clock.

-source_latency_included

Specifies whether the clock source latency should not be added to the input delay value. If this option is not specified, the clock source latency of the related clock is added to the input delay value. It has no effect if the input delay is not specified with respect to any clock.

delay_value

Specifies the path delay. The *delay_value* option must be in units consistent with the technology library used during analysis. The *delay_value* option represents the amount of time that the signal is available after a clock edge. This usually represents a combinational path delay from the clock pin of a register.

port_pin_list

Provides a list of input port or internal pin names in the current design to which *delay_value* is assigned. If you specify more than one object, the objects are enclosed in braces ({}).

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets input path delay values for the current design. Used with the **set_load** and **set_driving_cell** commands, the input and output delays characterize the operating environment of the current design.

The **set_input_delay** command sets input path delays on input ports relative to a clock edge. Unless specified, input ports are assumed to have zero input delay. For inout (bidirectional) ports, you can specify the path delays for both input and output modes.

To describe a path delay from a level-sensitive latch, use the *-level_sensitive* option. If the latch is positive-enabled, set the input delay relative to the rising clock edge. If the latch is negative enabled, set the input delay relative to the falling clock edge. If time is borrowed at that latch, add that time borrowed to the path delay from the latch when determining input delay.

- If you specify the **set_input_delay** command relative to a clock defined at the same port and the port has data sinks, the command is ignored and an error message is issued. There is only one signal coming to port, and it cannot be at the same time data relative to a clock and the clock signal itself.
- If you specify the **set_input_delay** command relative to a clock defined at a different port and the port has data sinks, the input delay is set and controls data edges launched from the port relative to the clock.
- Regardless of the location of the data port, if the clock port does not fanout to data sinks, the input delay on the clock port is ignored and you receive an error message.

  To control the clock source latency for any clocks defined on an input port, you must use the **set_clock_latency** command.

  To list input delays associated with ports, use the **report_port** command. To remove input delay values, use the **remove_input_delay** or **reset_design** command.

EXAMPLES

The following example sets an input delay of 2.3 for ports "IN1" and "IN2" on a combinational design. Because the design is combinational, no clock is needed.

```
ptc_shell> set_input_delay 2.3 { IN1 IN2 }
```

The following example sets input delay of 1.2 relative to the rising edge of "CLK1" for all input ports in the design.

```
ptc_shell> set_input_delay 1.2 -clock CLK1 [all_inputs]
```

The following example sets the input and output delays for the bidirectional port "INOUT1". The input signal arrives at "INOUT1" 2.5 units after the falling edge of "CLK1". The output signal is required at "INOUT1" at 1.4 units before the rising edge of "CLK2".

```
ptc_shell> set_input_delay 2.5 -clock CLK1 -clock_fall { INOUT1 }

ptc_shell> set_output_delay 1.4 -clock CLK2 { INOUT1 }
```

The following example models the situation where there are three paths to input port "IN1". The first path is relative to the rising edge of "CLK1". The second path is relative to the falling edge of "CLK1". The third path is relative to the falling edge of "CLK2". The **-add_delay** option is used to indicate that new input delay information does not cause old information to be removed.

```
ptc_shell> set_input_delay 2.2 -max clock CLK1 -add_delay { IN1 }

ptc_shell> set_input_delay 1.7 -max clock CLK1 -clock_fall -add_delay { IN1 }

ptc_shell> set_input_delay 4.3 -max clock CLK2 -clock_fall -add_delay { IN1 }
```

In the following example, two different maximum delays and two minimum delays for port "A" are specified using **-add_delay**. Since the information is relative to the same clock and clock edge, only the largest of the maximum values and the smallest of the minimum values are maintained (in this case, 5.0 and 1.1). If **-add_delay** is not used, the new information overwrites the old information.

```
ptc_shell> set_input_delay 3.4 -max -clock CLK1 -add_delay { A }

ptc_shell> set_input_delay 5.0 -max -clock CLK1 -add_delay { A }

ptc_shell> set_input_delay 1.1 -min -clock CLK1 -add_delay { A }

ptc_shell> set_input_delay 1.3 -min -clock CLK1 -add_delay { A }
```

SEE ALSO

```
all_inputs(2)
```

```
create_clock(2)
current_design(2)
remove_input_delay(2)
report_port(2)
reset_design(2)
set_driving_cell(2)
set_load(2)
set_output_delay(2)
```

set_input_transition

Sets a fixed transition time on input or inout ports.

SYNTAX

```
string set_input_transition [-rise]
   [-fall]
   [-min]
   [-max]
   [-clock clock_name]
   [-clock_fall]
   transition
   port_list

float transition
list port_list
```

ARGUMENTS

-rise

Sets rise transition only.

-fall

Sets fall transition only.

-min

Sets transition for minimum conditions.

-max

Sets transition for maximum conditions.

-clock *clock_name*

The input transition is set relative the specified clock.

-clock_fall

Specifies that the transition is relative to the falling edge of the clock. The default is the rising edge.

transition

Port transition value.This is a floating point number greater than or equal to zero.

port_list

A list of input or inout ports.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_input_transition** command specifies a fixed transition time for a list of input or inout ports. This transition time is not affected by the capacitance of the net connected to the port. The transition time calculates delays for nets and cells in the transitive fanout of the port. The port itself has no cell delay.

The transition time can be relative to a clock by using -clock option. This means the transition apply only those external paths driven by the clock. Using -clock or -clock_fall option can be meaningful only when timing_slew_propagation_mode is in worst_arrival mode. Note If it is in worst_slew mode the worst of input transitions specified with this command will be used, disreguard of the clock(s) specified. If a clock-relative input transition is set on a port, these transitions will not be accessible via **get_attribute** until -clock and -clock_fall options are supported for set_input_transition in SDC.

To display port transition or drive capability information, use **report_port -drive**.

There are two other methods of describing port drive capability. The **set_driving_cell** command causes the port to have transition time calculated as if a given library cell was driving the net. The **set_driving_cell** command also has a cell delay equal to the load-dependent portion of the delay driving the net of the library cell. The driving cell approach is accurate for nonlinear delay models even if the capacitance is changed. Another method is to use **set_drive_resistance**, which models the driver as a linear resistance.

EXAMPLES

This example specifies that ports matching the pattern "DATA_IN*" has a transition time of 0.75 units.

```
ptc_shell> set_input_transition 0.75 [get_ports DATA_IN*]
```

SEE ALSO

set_driving_cell(2)

set_drive_resistance(2)
report_port(2)
set_clock_transition(2)

set_input_units

This command specifies the units that are used for input.

SYNTAX

```
string set_input_units
   [-time_unit time_unit ]
   [-resistance_unit resistance_unit ]
   [-capacitance_unit capacitance_unit ]
   [-voltage_unit voltage_unit ]
   [-current_unit current_unit ]
   [-power_unit power_unit ]

string  time_unit
string  resistance_unit
string  capacitance_unit
string  voltage_unit
string  current_unit
string  power_unit
```

ARGUMENTS

-time_unit *time_unit*

Specifies the time unit for user input. This can be of the form "1ns", "10.0ps", or "1.0e-12".

-resistance_unit *resistance_unit*

Specifies the resistance unit for user input. This can be of the form "1kOhm", "100.0Ohm", or "1.0e4".

-capacitance_unit *capacitance_unit*

Specifies the capacitance unit for user input. This can be of the form "1pF", "100.0fF", or "1.0e-12".

-voltage_unit *voltage_unit*

Specifies the voltage unit for user input. This can be of the form "1V", "100.0mV", or "1.0".

-current_unit *current_unit*

Specifies the current unit for user input. This can be of the form "1mA", "1000.0uA", or "1.0".

-power_unit *power_unit*

Specifies the leakage power unit for user input. This can be of the form "1mW", "100.0mW", or "1.0".

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command specifies the units that are used for input. The tool maintains separate units for input and output. There are input unit values for time, resistance, capacitance, voltage, current, and power.

EXAMPLES

The following example sets the input units to 10ps, 1kOhm, 1pF, 1V, 1mA, and 1mW.

```
ptc_shell> set_input_units -time_unit 10ps -resistance_units 1kOhm \
 -capacitance_units 1pF -voltage_units 1V -current_units 1mA -power_units 1mW
```

SEE ALSO

get_input_unit(2)
get_output_unit(2)
set_output_units(2)

set_load

Sets the capacitance to a specified value on the specified ports and nets in the current design.

SYNTAX

```
Boolean set_load
   [-min]
   [-max]
   [-rise]
   [-fall]
   [-subtract_pin_load]
   [-pin_load]
   [-wire_load]
   capacitance
   objects

capacitance   float
objects       list
```

ARGUMENTS

-min

Indicates that the *capacitance* is the minimum capacitance. Applies only to designs in min-max mode (minimum and maximum operating conditions).

-max

Indicates that the *capacitance* is the maximum capacitance. Applies only to designs in min-max mode (minimum and maximum operating conditions).

-rise

Indicates that the *capacitance* is the rise capacitance. This option can be used in combination with **-min** or **-max**. Use this option only with ports. An error message is generated if the *objects* list contains nets.

-fall

Indicates that the *capacitance* is the fall capacitance. This option can be used in combination with **-min** or **-max**. Use this option only with ports. An error message is generated if the *objects* list contains nets.

-subtract_pin_load

Indicates that the current pin capacitances of the specified net are to be subtracted from *capacitance* before the net capacitance value is set. Any resulting negative net capacitance values are set to zero. With this option, the total capacitance computed on these nets during **update_timing** does not include pin capacitances. Use this option only if *capacitance* includes pin capacitances.

-pin_load

Indicates that the specified *capacitance* is a pin capacitance. Pin capacitance is not subject to "scaling" using tree_type. Use this option only with ports. An error message is generated if the *objects* list contains nets.

If you do not specify either the **-pin_load** or the **-wire_load** option, or both, **-pin_load** is the default.

-wire_load

Indicates that the specified *capacitance* is a wire capacitance. Pin capacitance is subject to "scaling" using tree_type. Use this option only with ports. An error message is generated if the *objects* list contains nets.

If you do not specify either the **-pin_load** or the **-wire_load** option, or both, **-pin_load** is the default.

capacitance

Specifies the capacitance value to be set on the ports and nets in *objects*. It is in the units of the technology library.

objects

Specifies a list of ports and nets in the current design, on which the *capacitance* is to be set. Note that if you specify a name (for example, *net_name*) instead of a real object_list (for example, by using the **get_port** command with the *port_name* option or the **get_net** command with the *net_name* option), the tool first searches the list of all ports and then searches the list of all nets for a match.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command sets the capacitance to a specified value on specified ports and nets in the current design. If the current design is hierarchical, you must link it using the **link** command. Depending on the options used, **set_load** sets these attributes on the specified objects:

```
wire_capacitance_max
wire_capacitance_min
pin_capacitance_max
pin_capacitance_min
```

If this command is issued without any options, the **pin_capacitance_max** attribute is set; in min-max mode, the **pin_capacitance_min** attribute is also set.

For ports, if **set_load** is issued with only the **-wire_load** option, then the **wire_capacitance_max** attribute is set on the specified ports; in min-max mode, the **wire_capacitance_min** attribute is also set.

However, the value is actually counted as part of the total wire capacitance and not as part of the pin/port capacitance.

In min-max mode, using either the **-min** or **-max** option sets only the ***_min** or ***_max** attributes, respectively.

If **-rise** is specifed, the **pin_capacitance_max_rise** and **pin_capacitance_min_rise** attributes are set. You can use this option in conjunction with **-min** or **-max** to set only one of the two attributes. The same conditions apply for the **-fall** option and the **pin_capacitance_*_fall** attributes.

The total capacitance on a net is the sum of all pin capacitances, port capacitances, and wire capacitances associated with that net. The specified *capacitance* overrides both the internally-estimated net capacitance value and any net capacitance set by the **read_parasitics** command.

You can also use the **set_load** command for nets at lower levels of the design hierarchy. These nets are specified in the "BLOCK1/BLOCK2/NET_NAME" format. The tool treats capacitances in such a way that timing on a subblock should be the same as timing on the higher-level design, since pin/wire caps on ports represent a part of the higher-level design.

To view capacitance values on ports and nets, use the **report_port** and **report_net** commands, respectively. To remove a capacitance value, use the **remove_capacitance** command; you can remove annotated capacitances from the full design by using the **reset_design** command.

EXAMPLES

The following example sets a capacitance of *2* units on the port named *the_answer*.

```
ptc_shell> set_load 2 the_answer
```

The following example sets a capacitance of *2.5* units (the estimated wire capacitance) plus the capacitance of three inverter input pins from the library named *tech_lib* on all output ports. It uses the user-defined **ptc_shell** variable **port_capacitance** to store this capacitance value.

```
pt-shell> set pin_cap \
?[get_attribute [get_lib_pins tech_lib/IV/A] pin_capacitance]
2.0
ptc_shell> set port_capacitance [expr 2.5 + (3 * $pin_cap)]
8.5
ptc_shell> set_load $port_capacitance [all_outputs]
1
```

The following example uses the **get_attribute** command to get the value of the **pin_capacitance** attribute on pin *Z* of *IV* from the *tech_lib* library and sets that value on the *input_1* and *input_2* ports.

```
ptc_shell> set_load \
?[get_attribute [get_lib_pins tech_lib/IV/Z] pin_capacitance] \
?[get_ports {input_1 input_2}]
```

The following example sets a capacitance of *3* on the **U1/U2/NET3** net. The wire capacitance is set to *3*. (Total net capacitance is 3 + the sum of the pin and port capacitances.)

```
ptc_shell> set_load 3 U1/U2/NET3
```

The following example sets a total net capacitance (wire capacitance + pin capacitances) of *3* on the

U1/U2/NET3 net. If the pin and port capacitances equal *2*, the wire capacitance is annotated with *1*; if the pin and port capacitances are *3* or more, the wire capacitance is annotated with *0*.

```
ptc_shell> set_load -subtract_pin_load 3 U1/U2/NET3
```

The following example sets a wire capacitance of *5* units on the port named *the_answer*.

```
ptc_shell> set_load -wire_load 5 the_answer
```

The following example removes the back-annotated capacitance on a port and on a net.

```
ptc_shell> remove_capacitance [get_ports the_answer]
ptc_shell> remove_capacitance [get_nets net12]
```

SEE ALSO

all_outputs(2)
current_design(2)
remove_capacitance(2)
report_net(2)
report_port(2)
reset_design(2)
set_drive(2)
set_wire_load(2)

set_max_area

Sets the **max_area** attribute on the current design to a specified value.

SYNTAX

```
int set_max_area [-ignore_tns]
   [-ignore_tns]
   area_value
```

Data Types

```
   area_value      float
```

ARGUMENTS

-ignore_tns

Specifies that the area is prioritized above total negative slack (TNS).

area_value

Specifies the value to which the maximum area should be set. The value must be greater than or equal to 0 (>= 0). The units of the *area_value* option must be consistent with the units in the technology library used during optimization.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_max_area** command sets the maximum area on the current design to the specified *area_value*. This attribute represents the target area of the design.

EXAMPLES

```
ptc_shell> set_max_area 250.0
1
```

SEE ALSO

current_design(2)
reset_deign(2)

set_max_capacitance

Sets maximum capacitance for pins, ports, clocks or designs.

SYNTAX

```
string set_max_capacitance
   capacitance_value
   [-clock_path]
   [-data_path]
   [-rise]
   [-fall]
   object_list
```

Data Types

capacitance_value	float
object_list	list

ARGUMENTS

-clock_path

Specifies all pins that are in the network of the specific clocks in the *object_list*.

-data_path

Specifies all pins that are in the data paths launched by the specific clocks in the *object_list*.

-rise

Specifies rising capacitance for all selected pins.

-fall

Specifies falling capacitance for all selected pins.

capacitance_value

Capacitance limit (Value >= 0). This is the maximum total capacitance (pin plus wire capacitance) in library capacitance units.

object_list

Provides a list of pins, ports, clocks or designs on which to set maximum capacitance.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies a maximum capacitance on pins, ports or design. If maximum capacitance is set on a pin or port, the net connected to that pin or port is expected to have a total capacitance less than the specified with the *capacitance_value* option. If specified on a design, the default maximum capacitance for that design is set. Library cell pins also can have a *max_capacitance* value specified.

If maximum capacitance is set on a clock, the maximum capacitance is applied to all pins in this specified clock domain. Within a clock domain, you can optionally restrict the constraint further to clock paths only or data paths only, and to rising or falling capacitance only. Note that the *clock_path*, *data_path*, *rise*, and *fall* options can only be used when the max capacitance limit is set on a clock and does not apply to design, pin, or port.

The most restrictive of the design limit, pin, clock, library, or port limit is used.

To remove maximum capacitance limits from designs or ports, use the **remove_max_capacitance** command.

EXAMPLES

The following example sets a maximum capacitance limit of 2.0 units on ports "OUT*".

```
ptc_shell> set_max_capacitance 2.0 [get_ports "OUT*"]
```

The following example sets the default maximum capacitance limit of 5.0 units on the current design.

```
ptc_shell> set_max_capacitance 5.0 [current_design]
```

The following example sets a maximum capacitance limit of 0.8 units on all pins of the clock network of CLK.

```
ptc_shell> set_max_capacitance 0.8 [get_clocks CLK] -clock_path
```

The following example sets a maximum capacitance limit of 0.7 units on all pins of the data path of CLK.

```
ptc_shell> set_max_capacitance 0.7 [get_clocks CLK] -data_path
```

SEE ALSO

```
current_design(2)
remove_max_capacitance(2)
```

set_max_delay

Specifies a maximum delay for timing paths.

SYNTAX

```
Boolean set_max_delay
   [-rise] [-fall]
   [-ignore_clock_latency]
   [-reset_path]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-comment comment_string]
   delay_value
```

Data Types

from_list	list
rise_from_list	list
fall_from_list	list
through_list	list
rise_through_list	list
fall_through_list	list
to_list	list
rise_to_list	list
fall_to_list	list
delay_value	float
comment_string	string

ARGUMENTS

-rise

Indicates that only rising path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-fall

Indicates that only falling path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-ignore_clock_latency

Indicates that launch and capture clock latencies are to be ignored when computing slack on the specified paths.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise or fall setup or hold type is reset. Using this option is equivalent to using the **reset_path** command with similar arguments before issuing **set_max_delay**.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, cells, and nets through which the paths must pass for maximum delay definition. Nets are interpreted to imply the net segment's local driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than one time in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-rise_through ***rise_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects. You can specify **-rise_through** more than one time in a single command invocation. For a discussion of multiple **-through**, **-rise_through**, and **-fall_through** options, see the DESCRIPTION section.

-fall_through ***fall_through_list***

This option is similar to the **-through** option, but applies only to paths with a falling transition at the

specified objects. You can specify **-fall_through** more than one time in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

delay_value

Specifies a floating point number that represents the required maximum delay value for specified paths. *delay_value* must have the same units as the logic library used during analysis. If a path startpoint is on a sequential device, clock skew is included in the computed delay. If a path startpoint has an input delay specified, that delay value is added to the path delay. If a path endpoint is on a sequential device, clock skew and library setup time are included in the computed delay. If the endpoint has an output delay specified, that delay is added into the path delay.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command specifies a required maximum delay for timing paths in the current design. The path length for any startpoint in *from_list* to any endpoint in *to_list* must be less than *delay_value*.

The value of a **max_rise_delay** attribute cannot be less than that of a **min_rise_delay** attribute on the same path (and similarly for fall attributes). If this condition occurs, the older attribute is removed.

Individual maximum delay targets are automatically derived from clock waveforms and port input or output

delays. For more information, refer to the **create_clock**, **set_input_delay**, and **set_output_delay** command man pages.

The **set_max_delay** command is a point-to-point timing exception command. For example, the command overrides the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include **set_multicycle_path**, **set_min_delay**, and **set_false_path**. A **set_max_delay** or **set_min_delay** command overrides a **set_multicycle_path** command.

The more general commands apply to more than one path. For example, either **-from** or **-to** is used (but not both), or clocks are used in the specification. Within a given point-to-point exception command, the more specific command overrides the more general. The following list of commands is arranged in order, from the highest to the lowest precedence (more specific to more general).

1. set_max_delay -from *pin* -to *pin*
2. set_max_delay -from *pin* -to *clock*
3. set_max_delay -from *pin*
4. set_max_delay -from *clock* -to *pin*
5. set_max_delay -to *pin*
6. set_max_delay -from *clock* -to *clock*
7. set_max_delay -from *clock*
8. set_max_delay -to *clock*

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

In some earlier releases of constraint consistency, a single **-through** option specifies multiple traversal points. You can revert to this behavior by setting the **timing_through_compatability** variable to true. If you do so, the behavior is as illustrated in the following example, which specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through {B1 C1} -to D1
```

In this mode, you cannot use more than one **-through** option in a single command.

To list the **max_delay**, **min_delay**, **multicycle_path**, and **false_path** information for the design, use the **report_exceptions** command.

To remove information set by **set_max_delay**, use the **reset_path** or **reset_design** command.

EXAMPLES

The following command specifies that any delay path to port "Y" must be less than 10.0 units.

```
ptc_shell> set_max_delay 10.0 -to {Y}
```

The following command specifies that all paths from ff1a or ff1b to ff2e must have delays less than 15.0 units.

```
ptc_shell> set_max_delay 15.0 -from {ff1a ff1b} -to {ff2e}
```

The following example specifies that all paths to endpoints clocked by PHI2 must have delays less than 8.5 units.

```
ptc_shell> set_max_delay 8.5 -to [get_clocks PHI2]
```

The following example sets a requirement that all paths leading to ports named "busA[*]" must have delays less than 5.0.

```
ptc_shell> set_max_delay 5.0 -to "busA[*]"
```

The following example specifies that all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} must have delays less than 8.0 units.

```
ptc_shell> set_max_delay 8.0 -from ff1/CP -through {U1/Z U2/Z} -through {U3/Z U4/C} -to ff2/D
```

The following example specifies that all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall through one or more of {U3/Z U4/C} must have delays less than 8.0 units.

```
ptc_shell> set_max_delay 8.0 -from ff1/CP -rise_through {U1/Z U2/Z} -fall_through {U3/Z U4/C} -to f
```

SEE ALSO

create_clock(2)
current_design(2)
group_path(2)
reset_design(2)
reset_path(2)
set_false_path(2)
set_input_delay(2)
set_min_delay(2)
set_multicycle_path(2)
set_output_delay(2)

set_max_fanout

Sets maximum fanout for input ports or designs.

SYNTAX

```
string set_max_fanout fanout_value
   object_list
```

Data Types

```
   fanout_value     float
   object_list      list
```

ARGUMENTS

fanout_value

Fanout limit (Value >= 0). This is the maximum fanout load in library fanout units.

object_list

Provides a list of input ports or designs on which to set maximum fanout.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies a maximum fanout on input ports or designs. If maximum fanout is set on a port, the net connected to that port is expected to have the **fanout_load** attribute less than the specified **fanout_value** attribute. Fanout load for a net is the sum of the **fanout_load** attributes for all input pins on the net. If specified on a design, the default maximum fanout for that design is set. Library cell pins may also have the **max_fanout** value specified. The most restrictive of the design limit and the pin or port limit will be used.

To remove maximum fanout limits from designs or ports, use the **remove_max_fanout** command.

EXAMPLES

The following example sets a maximum fanout limit of 2.0 units on ports "IN*".

```
ptc_shell> set_max_fanout 2.0 [ge_ports "IN*"]
```

The following example sets the default maximum fanout limit of 5.0 units on the current design.

```
ptc_shell> set_max_fanout 5.0 [current_design]
```

SEE ALSO

current_design(2)
remove_max_fanout(2)
get_ports(2)
set_fanout_load(2)

set_max_time_borrow

Limits time borrowing for latches.

SYNTAX

```
string set_max_time_borrow value
   object_list
```

Data Types

```
   value          float
   object_list    list
```

ARGUMENTS

value

Specifies the value to which the **max_time_borrow** attribute is set. Defines the desired limit of time borrowing on the latches specified in the *object_list* option. The *delay_value* option must be between zero and the default maximum derived from the waveform. By default, the maximum is derived from the ideal clock waveform driving each latch, and is equal to (closing_edge - open_edge). Library setup and data-to-Q propagation times are automatically taken into account. The *delay_value* option is in the same units as those in the technology library used during analysis.

object_list

Specifies a list of objects in the **current_design** command for which time borrowing is to be limited to the *value* option. The objects can be clocks, latch cells, data pins, or clock (enable) pins. If a cell is specified, all enabled pins on that cell are affected.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the **max_time_borrow** attribute to the *value* option to constrain the amount of time borrowing

possible for level-sensitive latches. To meet delay targets, the **set_max_time_borrow** command prevents automatic use of all or part of the enabling clock pulse on a latch.

If the **max_time_borrow** attribute is set on both data and clock (enable) pins of a level-sensitive latch, the value set on the data pin takes higher priority.

To get the **max_time_borrow** attributes on the design, use the **get_attribute** command.

To undo the **set_max_time_borrow** command, use the **remove_max_time_borrow** command.

EXAMPLES

The following example restricts time borrowing on latches **latch1a** and **latch2f** to 4.0 units.

```
ptc_shell> set_max_time_borrow 4.0 { latch1a latch2f }
```

The following example specifies that no time borrowing will take place on **latch1c**.

```
ptc_shell> set_max_time_borrow 0.0 { latch1c }
```

SEE ALSO

current_design(2)
remove_max_time_borrow(2)
get_attribute(2)

set_max_transition

Sets maximum transition for pins, ports, clocks or designs with respect to the main library trip-points.

SYNTAX

```
string set_max_transition
   transition_value
   [-clock_path][-data_path]
   [-rise][-fall]
   object_list
```

Data Types

```
   transition_value      float
   object_list           list
```

ARGUMENTS

-clock_path

Specifies all pins that are in the network of the specific clocks in the object_list.

-data_path

Specifies all pins that are in the data paths launched by the specific clocks in the *object_list* option.

-rise

Specifies rising transition for all selected pins.

-fall

Specifies falling transition for all selected pins.

transition_value

Sets the transition limit (Value >= 0). This is the maximum transition time in library time units.

object_list

Provides a list of pins, ports, clocks or designs on which to set maximum transition.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies a maximum transition on pins, ports or designs. If maximum transition is set on a pin or port, the pin or port is expected to have transition time less than the specified *transition_value* option. If specified on a design, the default maximum transition for that design is set. Library cell pins can also have a *max_transition* value specified. The most restrictive of the design limit and the pin or port limit is used. If user entered limit is the most restrictive limit, it is scaled to that of the pin trip-points during slack computation.

If maximum transition is set on a clock, the maximum transition is applied to all pins in this specified clock domain. Within a clock domain, you can optionally restrict the constraint further to clock paths only or data paths only, and to rising transitions only or falling transitions only.

The *-clock_path*, *-data_path*, *-rise*, and *-fall* options is used only if the list of objects is a clock list, not a pin or port list or design. If a clock list is specified without *-clock_path*, *-data_path*, *-rise* or *-fall*, both clock path and data path, both rise and fall are considered by default.

To remove maximum transition limits from designs or ports, use **remove_max_transition**.

EXAMPLES

The following example sets a maximum transition limit of 2.0 units on ports "OUT*".

```
ptc_shell> set_max_transition 2.0 [get_ports "OUT*"]
```

The following example sets the default maximum transition limit of 5.0 units on the current design.

```
ptc_shell> set_max_transition 5.0 [current_design]
```

The following example sets a maximum transition limit of 2.0 units on all pins in the clock network of CLK.

```
ptc_shell> set_max_transition 2.0 [get_clocks CLK] -clock_path
```

SEE ALSO

```
current_design(2)
remove_max_transition(2)
```

set_message_info

Sets some information about diagnostic messages.

SYNTAX

```
string set_message_info
   -id message_id
   [-limit max_limit]
   [-severity severity]
   [-stop_on]
```

Data Types

```
   message_id   string
   max_limit    integer
   severity     string
```

ARGUMENTS

-id *message_id*

Information is to be set for the given *message_id*. The message must exist.

-limit *max_limit*

Set the maximum number of occurrences for *message_id*. This is an integer greater than or equal to zero. If you set it to zero, that means the number of occurrences of the message is unlimited. Messages which occur after a limit is reached are automatically suppressed.

-severity *severity*

Set the message severity for *message_id*. You can use this option to change the default severity of any message to information, error or warning.

-stop_on

Forces a Tcl error if the message occurs.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_message_info** command sets constraints on diagnostic messages (typically error, warning, and informational messages).

You can set a upper limit for the number of occurrences of a message. You can set this to zero to indicate that there is no limit. You can retrieve the current limit for a message using the **get_message_info** ***command. When the limit is exceeded, all future occurrences of the message are automatically suppressed. A count of the total occurrences (including those suppressed) can be retrieved by using the get_message_info command.***

You can change the default severity of messages by using the -severity option. You can also change the message severity from previous user settings. When you change the severity, the output of the **report_constraint_analysis** ***command and the print_message_info command reflects the change and counts the previous occurrences of the message in the new severity section.***

EXAMPLES

The following example uses **set_message_info** to set a limit on the number of APP-027 messages to 100. When the 101st APP-027 message is about to be issued, you will be warned that the limit has been exceeded, and that all future occurrences will be suppressed.

```
shell> set_message_info -id APP-027 -limit 100
shell> do_command
Warning: can't find node U27.1 (APP-027)
Warning: can't find node U27.2 (APP-027)
Warning: can't find node U27.3 (APP-027)
...
Warning: can't find node U27.100 (APP-027)
Note - message 'APP-027' limit (100) exceeded.  Remainder will be suppressed.
1
shell>
```

SEE ALSO

```
get_message_info(2)
print_message_info(2)
suppress_message(2)
```

set_min_capacitance

Sets minimum capacitance for ports or designs.

SYNTAX

```
string set_min_capacitance
   capacitance_value
   object_list
```

Data Types

```
   capacitance_value float
   object_list       list
```

ARGUMENTS

capacitance_value

Sets capacitance limit (Value >= 0). This is the minimum total capacitance (pin plus wire capacitance) in library capacitance units.

object_list

Provides a list of input or inout ports or designs on which to set minimum capacitance.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies a minimum capacitance on input/inout ports or designs. If minimum capacitance is set on a port, the net connected to that port is expected to have total capacitance greater than the specified *capacitance_value*. If specified on a design, the default minimum capacitance for that design is set. Library cell pins can also have a **min_capacitance** value specified. The most restrictive of the design limit and the pin or port limit is used.

To remove minimum capacitance limits from designs or ports, use **remove_min_capacitance**.

EXAMPLES

The following example sets a minimum capacitance limit of 0.2 units on ports IN*.

```
pt_shell> set_min_capacitance 0.2 [get_ports "IN*"]
```

The following example sets the default minimum capacitance limit of 0.1 units on the current design.

```
pt_shell> set_min_capacitance 0.1 [current_design]
```

SEE ALSO

current_design(2)
remove_min_capacitance(2)

set_min_delay

Specifies a minimum delay for timing paths.

SYNTAX

```
Boolean set_min_delay
   [-rise] [-fall]
   [-ignore_clock_latency]
   [-reset_path]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-comment comment_string]
   delay_value
```

Data Types

from_list	list
rise_from_list	list
fall_from_list	list
through_list	list
rise_through_list	list
fall_through_list	list
to_list	list
rise_to_list	list
fall_to_list	list
delay_value	float
comment_string	string

ARGUMENTS

-rise

Indicates that only rising path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-fall

Indicates that only falling path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-ignore_clock_latency

Indicates that launch and capture clock latencies are to be ignored when computing slack on the specified paths.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise or fall setup or hold type is reset. Using this option is equivalent to using the **reset_path** command with similar arguments before issuing **set_min_delay**.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, cells, and nets through which the paths must pass for maximum delay definition. Nets are interpreted to imply the net segment's local driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than one time in one command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-rise_through ***rise_through_list***

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects. You can specify **-rise_through** more than one time in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-fall_through ***fall_through_list***

This option is similar to the **-through** option, but applies only to paths with a falling transition at the

specified objects. You can specify **-fall_through** more than one time in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

delay_value

Specifies a floating point number that represents the required minimum delay value for specified paths. *delay_value* must have the same units as the logic library used during analysis. If a path startpoint is on a sequential device, clock skew is included in the computed delay. If a path startpoint has an input external delay specified, that delay value is added to the path delay. If a path endpoint is on a sequential device, clock skew and library setup time are included in the computed delay. If the endpoint has an output external delay specified, that delay is added into the path delay.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command specifies a required minimum delay for timing paths in the current design. The path length for any startpoint in *from_list* to any endpoint in *to_list* must be greater *delay_value*.

Use **set_min_delay** in the following two cases:

1. To set a target minimum delay for output ports in a combinational design.

2. To override the default single cycle timing for paths, where **set_multicycle_path** is not sufficient.

The value of a **min_rise_delay** attribute cannot be greater than that of a **max_rise_delay** attribute for the same path (and similarly for fall attributes). If this condition occurs, the older attribute is removed.

Individual minimum delay targets are automatically derived from clock waveforms and port input or output delays. For more information, refer to the **create_clock**, **set_input_delay**, and **set_output_delay** command man pages.

The **set_min_delay** command is a point-to-point timing exception command; it overrides the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include **set_multicycle_path**, **set_max_delay**, and **set_false_path**. A **set_max_delay** or **set_min_delay** command overrides a **set_multicycle_path** command.

The more general commands apply to more than one path; either **-from** or **-to** is used (but not both), or clocks are used in the specification. Within a given point-to-point exception command, the more specific command overrides the more general. The following list of commands is arranged in order, from highest to lowest precedence (more specific to more general):

1. set_min_delay -from *pin* -to *pin*
2. set_min_delay -from *pin* -to *clock*
3. set_min_delay -from *pin*
4. set_min_delay -from *clock* -to *pin*
5. set_min_delay -to *pin*
6. set_min_delay -from *clock* -to *clock*
7. set_min_delay -from *clock*
8. set_min_delay -to *clock*

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

In some earlier releases of constraint consistency, a single **-through** option specifies multiple traversal points. You can revert to this behavior by setting the **timing_through_compatability** variable to true. If you do so, the behavior is as illustrated in the following example, which specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through {B1 C1} -to D1
```

In this mode, you cannot use more than one **-through** option in a single command.

To remove information set by **set_min_delay**, use the **reset_path** or **reset_design** command.

EXAMPLES

The following command specifies that any delay path to port "Y" must be greater than 12.5 units.

```
ptc_shell> set_min_delay 12.5 -to Y

The following command specifies that all paths from A1 and A2 to Z5 must
have delays greater than 4.0 units.

ptc_shell> set_min_delay 4.0 -from {A1 A2} -to Z5
```

The following example specifies that all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} must have delays greater than 3.0 units.

```
ptc_shell> set_min_delay 3.0 -from ff1/CP -through {U1/Z U2/Z} -through {U3/Z U4/C} -to ff2/D
```

The following example specifies that all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall through one or more of {U3/Z U4/C} must have delays greater than 3.0 units.

```
ptc_shell> set_min_delay 3.0 -from ff1/CP -rise_through {U1/Z U2/Z} -fall_through {U3/Z U4/C} -to f
```

SEE ALSO

current_design(2)
reset_design(2)
set_input_delay(2)
set_output_delay(2)
reset_path(2)
set_false_path(2)
set_multicycle_path(2)
set_max_delay(2)

set_min_pulse_width

Sets a minimum pulse width constraint for specified design objects.

SYNTAX

```
string set_min_pulse_width
   [-low]
   [-high]
   value
   [object_list]
```

Data Types

```
   value              float
   object_list        list
```

ARGUMENTS

-low

Indicates that the minimum pulse width constraint specified by *value* is to apply only to low clock signal levels. If you do not specify the *-low* or *-high* option, both low and high pulses of clock signals are constrained.

-high

Indicates that the minimum pulse width constraint specified by *value* is to apply only to high clock signal levels. If you do not specify the *-low* or *-high* option, both low and high pulses of clock signals are constrained.

value

A positive floating point value that specifies the minimum pulse width check to be applied to clock signals.

object_list

Specifies a list of clocks, cells, pins, or ports in the current design, to which the minimum pulse width check is to be applied. If you specify a cell or clock, all pins on the cell or clock are affected. If you do not specify any objects, the minimum pulse width check is applied to the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_min_pulse_width** command specifies a minimum pulse width check to be applied to clock signals in the clock tree or at sequential devices. This value overrides the default values for clock tree minimum pulse width checks. Use this command for sequential clock pin pulse width checks, to add a more restrictive value than the one specified in library.

A clock pulse width problem occurs if the negation of the clock signal happens too soon after the assertion of the clock signal. Clock pulse width violations can cause the following problems:

- Sequential devices (flip-flops and latches) might not capture data properly.
- In some logic circuits, the entire pulse could disappear.

Two types of minimum pulse width checks are performed:

- Clock Pulse Width Check at Sequential Devices: This check is performed on clock pins of sequential elements. This check verifies that a minimum separation exists between the trailing edge and leading edge of the clock that is clocking the sequential elements. The library defines the constraints, which are environmentally scalable. The **set_min_pulse_width** command overrides the library value. The most restrictive value of pulse width (library-specified or user-asserted) is used. No default value is assumed.

- Clock Tree Pulse Width Check at Logic Circuits: The clock tree pulse width check ensures that the clock pulse is propagated reliably through the logic. This check is performed on the combinational logic circuits through which the clock signals are propagated. No default is assumed for the clock tree pulse width check. The **set_min_pulse_width** command overrides the library value.

The **reset_design** command removes all user-specified attributes from a design, including those set by the **set_min_pulse_width**.

EXAMPLES

The following example sets a minimum pulse width requirement of 2.0 for both low and high pulses of clock signals.

```
pt_shell> set_min_pulse_width 2.0 [get_clocks CK1]
```

The following example sets a minimum pulse width requirement of 2.5 for the low pulse of the clock signal on the pin U1/Z.

```
pt_shell> set_min_pulse_width -low 2.5 U1/Z
```

SEE ALSO

```
current_design(2)
remove_min_pulse_width(2)
```

set_mode

Selects the active mode of cell mode groups.

SYNTAX

```
string set_mode
   [-type cell | design]
   [mode_list]
   [instance_list]
```

Data Types

```
   mode_list          list
   instance_list      list
```

ARGUMENTS

-type cell | design

Indicates the type of mode to be made active. This option has the following mutually-exclusive valid values: **design** and **cell**. The **cell** value specifies that cell modes are to be made active. Cell modes are defined on library cells in the library. The **design** value specifies that design modes are to be made active. Design modes are now obsolete and this option will be removed in a future release. If the **-type** value is omitted from the command, then this is equivalent to specifying the **-type cell** value.

mode_list

Specifies a list of modes, each of which is to be made the active mode for its mode group. If **-type** has a cell value, then the *mode_list* option must contain only cell modes.

instance_list

Specifies a list of instances for which the specified cell modes are made active. This list must only be included with the **-type cell** value.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Selects the active mode for a mode group or for several mode groups and disables modes in the same group as the selected active mode. Mode groups must either have all modes enabled (default setting) or have one of their modes enabled and all others disabled. This command sets cell modes only. Design modes are not supported.

Cell mode groups are defined in the library. Each library cell can have a set of cell mode groups. Each of these cell mode groups can have two or more cell modes. Each of these cell modes can be mapped to a set of timing arcs of the library cell.

When a cell mode is made active for a given instance of the library cell, the cell mode is enabled and all of its timing arcs are enabled for that cell. All other cell modes are automatically disabled. The timing arcs of the disabled cell modes are then automatically disabled. To view what modes have been enabled or disabled, use the **report_mode -type** cell. To view what arcs have been disabled, use the **report_disable_timing** command.

In the special case of an arc having multiple cell modes mapped to it, the arc is enabled if any of the modes are enabled.

Cell modes can be made active in two different ways, using the **set_mode -type cell** command, or through evaluation of mode conditions. When conflict arises, **set_mode** -type cell takes precedence over evaluation of mode conditions.

EXAMPLES

The following command directly selects the READ cell mode as the active mode for the RAM instance Uram1.

```
ptc_shell> set_mode READ Uram1
```

The following command is equivalent to the previous command.

```
ptc_shell> set_mode -type cell READ Uram1
```

SEE ALSO

reset_mode(2)
report_mode(2)

set_multicycle_path

Defines the multicycle path.

SYNTAX

```
Boolean set_multicycle_path
   [-setup] [-hold]
   [-rise] [-fall]
   [-start] [-end]
   [-reset_path]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]*
   [-rise_through rise_through_list]*
   [-fall_through fall_through_list]*
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-comment comment_string]
   path_multiplier

list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
int path_multiplier
string  comment_string
```

ARGUMENTS

-setup

Indicates that setup (maximum delay) calculations are to use the specified *path_multiplier*. Note that changing the path_multiplier for setup affects the default hold check as well. If neither **-setup** nor **-hold** is specified, *path_multiplier* is used for setup calculations and 0 is used for hold calculations.

-hold

Indicates that hold (minimum delay) calculations are to use the specified *path_multiplier*. Note that changing the path_multiplier for setup affects the default hold check as well. If neither **-setup** nor **-hold** is specified, *path_multiplier* is used for setup calculations and 0 is used for hold calculations.

-rise

Indicates that only rising path delays are to use *path_multiplier*. If neither **-rise** nor **-fall** is specified, both rising and falling delays are affected. Rise refers to a rising value at the path endpoint.

-fall

Indicates that only falling path delays are to use *path_multiplier*. If neither **-rise** nor **-fall** is specified, both rising and falling delays are affected. Fall refers to a falling value at the path endpoint.

-start

Indicates that the multicycle information is relative to the period of the start clock. The **-start** and **-end** options are needed only for multifrequency designs; otherwise, start and end are equivalent. The start clock is the clock source related to the register or primary input at the path startpoint.

The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with **-start** moves the relation backward one cycle of the start clock. A hold multiplier of 1 with **-start** moves the relation forward one cycle of the start clock.

-end

Indicates that the multicycle information is relative to the period of the end clock. The **-start** and **-end** options are needed only for multifrequency designs; otherwise, start and end are equivalent. The end clock is the clock source related to the register or primary output at the path endpoint.

The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with **-end** moves the relation forward one cycle of the end clock. A hold multiplier of 1 with **-end** moves the relation backward one cycle of the end clock.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise/fall setup/hold type is reset. Using this option is equivalent to using the **reset_path** command with similar arguments before issuing **set_multicycle_path**.

-from ***from_list***

Specifies a list of timing path startpoint objects. A vaild timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified.If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through ***through_list***

Specifies a list of pins, ports, cells and nets through which the multicycle paths must pass. Nets are interpreted to imply the net segment's local driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected. You can specify **-through** more than once in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-rise_through ***rise_through_list***

This option is similiar to the **-through** option, but applies only to paths with a rising transition at the specified objects. You can specify **-rise_through** more than once in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-fall_through ***fall_through_list***

This option is similiar to the **-through** option, but applies only to paths with a fall transition at the specified objects. You can specify **-fall_through** more than once in a single command invocation. For a discussion of multiple **-through**, **rise_through**, and **fall_through** options, see the DESCRIPTION section.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to ***fall_to_list***

Same as the **-to** option,but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-comment ***comment_string***

Associate a string description with the command for tracking purposes. This can be useful when writing out SDC to a tag, such as the methodology or tool that originally synthesized the command. This option is currently ignored in constraint consistency.

path_multiplier

An integer that specifies the number of cycles the data path must have for setup or hold, relative to the startpoint or endpoint clock, before data is required at the endpoint. For example, specifying a *path_multiplier* of 2 for setup implies a 2-cycle data path. If you use **-setup**, *path_multiplier* is applied to setup path calculations. If you use **-hold**, *path_multiplier* is applied to hold path calculations. If you do not specify either **-setup** or **-hold**, *path_multiplier* is used for setup and 0 is used for hold. Note that changing the multiplier for setup affects the hold check as well.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command specifies the number of cycles that the data path must have for setup or hold, so that that designated timing paths in the current design have no default setup or hold relations.

Constraint consistency applies certain rules to determine single cycle timing relationships for paths between clocked elements; the rules are based on active edges. For flip-flops, a single active edge both launches and captures data. For latches, the open edge launches data and the close edge latches data.

The setup check ensures that the correct data signal is available on destination registers in time to be correctly latched. The default setup behavior, called single-cycle setup, is as follows:

For multifrequency designs, there can be multiple setup relations between two clocks. For every latch edge of the destination clock, the setup check determines the nearest launch edge that precedes each latch edge. The smallest value of (setup_latch_edge - setup_launch_edge) determines the maximum delay requirement for this path.

You can override this default relationship by applying **set_multicycle_path** or **set_max_delay** to clocks, pins, ports, or cells. For example, setting the setup path multiplier to 2 with the **set_multicycle_path** command delays the latch edge one clock pulse. Note that changing the setup multiplier also affects the default hold check.

Most often, the setup check moves relative to the end clock. This move changes the data latch time at the path endpoint. In multi-frequency designs, use **set_multicycle_path -setup -start** to move the data launch time backward. The hold check ensures that data from the source clock edge that follows the setup launch edge is not latched by the setup latch edge, and also ensures that data from the setup launch edge is not latched by the destination clock edge that precedes the setup latch edge.

The hold check is determined relative to each valid setup relationship, after applying multicycle path multipliers. The most restrictive (largest) difference of (hold_latch_edge - hold_launch_edge) is used as a minimum delay requirement for the path.

Important Note: The hold relation is conservative. In some cases you need to override the default hold relation. For multifrequency designs, it is more straightforward to use the **set_min_delay** command to override the default instead of using **set_multicycle_path -hold**.

There are separate setup and hold multipliers for rise and fall. In most cases, these are set to the same value. You can use the **-rise** or **-fall** option to apply different values.

The **set_multicycle_path** command is a point-to-point timing exception command. The command can override the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include **set_max_delay**, **set_min_delay**, and **set_false_path**. False path

information always takes precedence over multicycle path information. A specific **set_max_delay** or set_min_delay command overrides a general **set_multicycle_path** command.

The more general commands apply to more than one path; either **-from** or **-to** (but not both), or clocks are used in the specification. Within a given point-to-point exception command, the more specific command overrides the more general. The following lists commands from highest to lowest precedence (more specific to more general):

1. set_multicycle_path -from *pin* -to *pin*
2. set_multicycle_path -from *pin* -to *clock*
3. set_multicycle_path -from *pin*
4. set_multicycle_path -from *clock* -to *pin*
5. set_multicycle_path -to *pin*
6. set_multicycle_path -from *clock* -to *clock*
7. set_multicycle_path -from *clock*
8. set_multicycle_path -to *clock*

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

In some earlier releases of constraint consistency, a single **-through** option specifies multiple traversal points. You can revert to this behavior by setting the **timing_through_compatability** variable to true. If you do so, the behavior is as illustrated in the following example, which specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through {B1 C1} -to D1
```

In this mode, you cannot use more than one **-through** option in a single command.

To undo a **set_multicycle_path** command, use the **reset_path** command. The **reset_design** command removes all attributes from the design, including those set by **set_multicycle_path**.

To disable setup and hold calculations for paths, use **set_false_path**. To list the point-to-point exceptions on a design, use **report_exceptions**.

EXAMPLES

The following example sets all paths between latch1b and latch2d to 2-cycle paths for setup. Hold is measured at the previous edge of the clock at latch2d.

ptc_shell> **set_multicycle_path 2 -from { latch1b } -to { latch2d }**

The following example moves the hold check to the preceding edge of the start clock.

ptc_shell> **set_multicycle_path -1 -from { latch1b } -to { latch2d }**

The following example is a two-phase level-sensitive design, where the designer expects paths from phi1 to phi1 to be zero cycles.

ptc_shell> **set_multicycle_path 0 -from phi1 -to phi1**

The following example uses **-start** to specify a 3-cycle path relative to the clock at the path startpoint. Clock sources are specified, affecting all sequential elements clocked by that clock, or ports with input or output delay relative to that clock.

```
ptc_shell> set_multicycle_path 3 -start -from { clk50mhz } -to { clk10mhz }
```

The following example sets all rising paths to ff12/D to 2 cycles, and falling paths to 1 cycle.

ptc_shell> **set_multicycle_path 2 -rise -to { ff12/D }**

ptc_shell> **set_multicycle_path 1 -fall -to { ff12/D }**

The following multifrequency example shows a 20ns clock to 10ns clock with multicycle path of 2. Assume a path from ff1 (clocked by CLK2) to ff2 (clocked by CLK1).

```
ptc_shell> create_clock -period 20 -waveform {0 10} CLK2
ptc_shell> create_clock -period 10 -waveform {0 5} CLK1
ptc_shell> set_multicycle_path 2 -setup -from ff1/CP -to ff2/D
```

The single-cycle setup relation is from the CLK1 edge at 0ns to the CLK2 edge at 10ns. With the multicycle path, the setup relation is now 20ns (from CLK1 edge at 0ns to CLK2 edge at 20ns). The hold relations are determined according to that setup relation.

1. Data from the source clock edge that follows the setup launch edge must not be latched by the setup latch edge. This implies a hold relation of 0ns (from CLK1 edge at 20ns to CLK2 edge at 20ns).

2. Data from the setup launch edge must not be latched by the destination clock edge that precedes the setup latch edge. This implies a hold relation of 10ns (from CLK1 edge at 0ns to CLK2 edge at 10ns).

The most restrictive (largest) hold relation is used, so the minimum delay requirement for this path is 10ns. This is a conservative check which might not apply to certain designs. Often you know that the destination register is disabled for the clock edge at 10ns. You can modify this default hold relation with the following to get an 0ns requirement.

ptc_shell> **set_multicycle_path 1 -hold -end -from ff1/CP -to ff2/D**

However, a simpler approach is to use the following:

ptc_shell> **set_min_delay 0 -from ff1/CP -to ff2/D**

The following example sets all timing paths from ff1/CP to ff2/D which passes through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} to 2 cycle paths for setup.

ptc_shell> **set_multicycle_path 2 -from ff1/CP -through {U1/Z U2/Z} -through {U3/Z U4/C} -to ff2/D**

SEE ALSO

current_design(2)
report_exceptions(2)
reset_design(2)
reset_path(2)
set_false_path(2)
set_max_delay(2)
set_min_delay(2)

set_operating_conditions

Defines the operating conditions (or environmental characteristics) for the *current design*.

SYNTAX

```
int set_operating_conditions
   [-analysis_type single | bc_wc | on_chip_variation]
   [-library lib]
   [condition]
   [-min min_condition]
   [-max max_condition]
   [-min_library min_lib]
   [-max_library max_lib]
   [-object_list objects]

string lib
string condition
string min_condition
string max_condition
string min_lib
string max_lib
list objects
```

ARGUMENTS

-analysis_type single | bc_wc | on_chip_variation

Indicates how the specified operating conditions are to be used. This option has the following mutually-exclusive valid values: **single**, **bc_wc**, and **on_chip_variation**. The **single** value specifies that only one operating condition is to be used. Specifying either **bc_wc** or **on_chip_variation** switches the design to min_max mode. The **bc_wc** value specifies that the min and max operating conditions are two extreme operating conditions. In the **bc_wc** analysis, setup violations are checked only for the maximum operating condition, and the hold violations are checked only for the minimum operating condition. The **on_chip_variation** value specifies that the minimum and maximum operating conditions represent, respectively, the lower and upper bounds of the maximum variation of operating conditions on the chip. All maximum path delays use the maximum operating condition, and all minimum path delays use the minimum operating condition.

-library *lib*

Specifies the library that contains definitions of the environmental characteristics to be used. This is

either a library name or a collection object.

condition

Specifies the name of the single operating condition.

-min ***min_condition***

Specifies the minimum operating condition used for checking minimum delays and hold violations. If you use this option you must also use the **-max** option.

-max ***max_condition***

Specifies the maximum operating condition used for checking maximum delays and setup violations. If you use this option you must also use the **-min** option.

-min_library ***min_lib***

Specifies the library that contains the *min_condition*. This is either a library name or a collection object. If you use this option you must also use the **-min** option.

-max_library ***max_lib***

Specifies the library that contains the *max_condition*. This is either a library name or a collection object. If you use this option you must also use the **-max** option.

-object_list ***objects***

Specifies the cells or ports on which to set operating conditions. If you do not use this option, operating conditions are set on the design. This option accepts both leaf cells and hierarchical blocks. You cannot use this option with the **-analysis_type** option.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Defines the operating conditions (or environmental characteristics) under which the *current design* is timed. You are in min_max mode if you use the **-min** and **-max** options. In that case, the default analysis type is **bc_wc**.

The specified operating conditions must be defined in one of the following: in *min_lib* (for the *min_condition* only), in *max_lib* (for the *max_condition* only), or in one of the libraries in the *link_path*. The order for a library search is as follows:

1. *lib*
2. *min_lib* or *max_lib*
3. *link_path*

If no operating conditions are specified for a design, the tool uses the default operating condition of the library to which the cell is linked. If the library does not have default operating conditions, no operating conditions are used.

Operating conditions set by using the **-object_list** option override the operating conditions set on design

or higher levels of hierarchy.

To view the operating conditions defined for the *current design* and to determine the libraries to which the *current design* is linked, use the **report_design** command. To view the operating conditions defined in the specified library, use the **report_lib** command. To view the operating conditions set on the cells or blocks library, use the **report_cell** command.

To remove operating conditions from the *current design*, use the **remove_operating_conditions** or the **reset_design** command.

EXAMPLES

The following example shows an operating condition definition, as it appears in the source text of a library.

```
operating_conditions("BCCOM") {
    process : 0.6 ;
    temperature : 20 ;
    voltage : 5.25 ;
    tree_type : "best_case_tree" ;
}
```

The name of this set of operating conditions is *BCCOM*. The parameters are defined as follows:

process \(em A floating-point number that represents the characteristics of a semiconductor manufacturing process.

temp \(em A floating-point number that represents the temperature of the defined environment.

voltage \(em A floating-point number that defines the upper boundary of the voltage range in which the defined environment operates. The lower boundary is always 0.0.

tree-type \(em The interconnect model for the environment. The tool uses the interconnect model to select a formula for calculating interconnect delays. Three models are available: **best_case_tree** that uses a lumped RC model, **worst_case_tree** in which all loads assume full wire resistance, and **balanced_tree** in which all loads share the wire resistance evenly.

When process factor, operating temperature, and operating voltage deviate from their nominal values, the tool uses a linear model to compensate for the effect of deviations on such values as cell delays, input loads, and output drives. The nominal values used are defined in the same library that contains the definitions of the set of operating conditions.

The following example uses operating condition WCIND found in the my_lib.db library to set the operating conditions to *WCIND* if the *link_path* is *my_lib.db*.

```
ptc_shell> set_operating_conditions WCIND
```

The following example uses operating condition WCIND found in the other_lib.db library to set the operating conditions to *WCIND* if the *link_path* is *other_lib.db*.

```
ptc_shell> set_operating_conditions WCIND -library other_lib.db
```

In the following example, the *BCCOM* values are used for minimum operating conditions and the *WCCOM* values are used for maximum operating conditions. When reporting maximum delays, the delays are

computed for the maximum operating conditions. When reporting minimum delays, the delays are computed for the minimum operating condition.

```
ptc_shell> set_operating_conditions -min BCCOM -max WCCOM \
-analysis_type bc_wc
```

SEE ALSO

create_operating_conditions(2)
remove_operating_conditions(2)
report_cell(2)
report_design(2)
report_lib(2)
reset_design(2)
default_oc_per_lib(3)

set_output_delay

Sets output path delay values for the current design.

SYNTAX

```
string set_output_delay [-clock clock_name]
   [-reference_pin pin_port_name]
   [-clock_fall]
   [-level_sensitive]
   [-rise]
   [-fall]
   [-max]
   [-min]
   [-add_delay]
   [-network_latency_included]
   [-source_latency_included]
   [-group_path group_name]
   delay_value
   port_pin_list
```

Data Types

```
   clock_name      list
   pin_port_name   list
   group_name      string
   delay_value     float
   port_pin_list   list
```

ARGUMENTS

-clock *clock_name*

Specifies the name of a clock to which the specified delay is to be related. If you specify **-clock_fall**, you must also specify **-clock**.

-reference_pin *pin_port_name*

Specifies the clock pin or port to which the specified delay is related. If you use this option, and if propagated clocking is being used, the delay value is related to the arrival time at the specified reference pin, which is clock source latency plus its network latency from the clock source to this reference pin. The options **-network_latency_included** and **-source_latency_included** cannot be used at the same time as the **-reference_pin** option. For ideal clock network, only source latency is

applied.

The pin specified with the **-reference_pin** option should be a leaf pin or port in a clock network, in the direct or transitive fanout of a clock source specified with the **-clock** option. If multiple clocks reach the port or pin where you are setting the input delay, and if the **-clock** option is not used, the command considers all of the clocks.

-clock_fall

Indicates that the delay is relative to the falling edge of the clock. If you specify **-clock_fall** with **-reference_pin**, the delay is relative to the falling transition of the reference pin. If you specify **-clock**, the default is the rising edge or the rising transition of the reference pin. If you specify **-clock_fall**, you must also specify **-clock**.

-level_sensitive

Indicates that the destination of the delay is a level-sensitive latch. This allows the tool to derive a setup and hold relationship for paths to this port as if it were a level-sensitive latch. If you do not use **-level_sensitive**, the output delay is treated as if it were a path to a flip-flop.

-rise

Indicates that *delay_value* refers to a rising transition on specified ports of the current design. If you do not specify **-rise** or **-fall**, rising and falling delays are assumed to be equal.

-fall

Indicates that *delay_value* refers to a falling transition on specified ports of the current design. If you do not specify **-rise** or **-fall**, rising and falling delays are assumed to be equal.

-max

Indicates that *delay_value* refers to the longest path. If you do not specify **-max** or **-min**, maximum and minimum output delays are assumed to be equal.

-min

Indicates that *delay_value* refers to the shortest path. If you do not specify **-max** or **-min**, maximum and minimum output delays are assumed to be equal.

-add_delay

Indicates that delay information is to be added to the existing output delay, instead of overwriting the output delay. Using **-add_delay**, you can capture information about multiple paths leading from an output port that are relative to different clocks or clock edges. For example, **set_output_delay 5.0 -max -rise -clock phi1 {OUT1}** removes all other maximum rise output delays from **OUT1**, because **-add_delay** is not specified. Other output delays with a different clock or with **-clock_fall** are removed.

In another example, **-add_delay** is specified: **set_output_delay 5.0 -max -rise -clock phi1 -add_delay {Z}**. If there is an output maximum rise delay for Z relative to the clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is a maximum rise output delay relative to a different clock or different edge of the same clock, it remains with the new delay.

-network_latency_included

Specifies whether the clock network latency should not be added to the output delay value. If this option is not specified, the clock network latency of the related clock is added to the output delay

value. It has no effect if the clock is propagated or the output delay is not specified with respect to any clock.

-source_latency_included

Specifies whether the clock source latency should not be added to the output delay value. If this option is not specified, the clock source latency of the related clock is added to the output delay value. It has no effect if the output delay is not specified with respect to any clock.

-group_path ***group_name***

Specifies the name of a group into which paths ending at the specified ports or pins are to be added. If the group does not already exist, one is created. This is equivalent to specifying the separate command **group_path -name** *group_name* **-to** *port_pin_list* in addition to **set_output_delay**. If you do not specify **-group_path**, the existing path grouping does not change.

delay_value

Specifies the path delay in units consistent with the technology library used during analysis. The *delay_value* represents the amount of time before a clock edge that the signal is required. For maximum output delay, this represents a combinational path delay to a register plus the library setup time of that register. For minimum output delay, this value is usually the shortest path delay to a register minus the library hold time.

port_pin_list

Specifies a list of output port or internal pin names in the current design to which *delay_value* is assigned. If more than one object is specified, the objects are enclosed in braces ({}).

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command sets output path delay values for the current design. The input and output delays characterize the operating environment of the current design when used with **set_load** and **set_driving_cell**.

The **set_output_delay** command sets output path delays on output ports relative to a clock edge. Output ports have no output delay unless specified. For inout (bidirectional) ports, you can specify the path delays for both input and output modes.

To describe a path delay to a level-sensitive latch, use the **-level_sensitive** option. If the latch is positive-enabled, set the output delay relative to the rising clock edge; if it is negative-enabled, set the output delay relative to the falling clock edge. If time is borrowed at that latch, subtract that time from the path delay to the latch when determining output delay.

Constraint consistency adds input delay to path delay for paths starting at primary inputs and to output delay for paths ending at primary outputs.

The **-group_path** option modifies the path grouping. Path grouping affects the maximum delay cost. The worst violator within each group adds to the cost.

If a reference pin is not reachable by any given clock, zero arrival time is assumed. If no clock is specified with a reference pin and this reference pin is not reachable by any clock, an unconstrained path is reported. This behavior is changed after X0506. The new behavior is these invalid constraints is ignored and path is constrained by whatever it should be as if no such constraints are defined at all. The **check_timing** command generates a warning if the reference pin is not reachable by any active clock, or if multiple clocks reach the reference pin and no clock is specified in the command.

To list output delays associated with ports, use **report_port**.

To remove output delay values, use **remove_output_delay** or **reset_design**. To modify paths grouped with the **-group_path** option, use the **group_path** command to place the paths in another group or the default group.

EXAMPLES

The following example sets an output delay of 1.7 relative to the rising edge of CLK1 for all output ports in the design.

```
ptc_shell> set_output_delay 1.7 -clock CLK1 [ all_outputs ]


The following example sets the input and output delays for the
bidirectional port INOUT1. The input signal arrives at INOUT1 2.5
units after the falling edge of CLK1. The output signal is required
at INOUT1 at 1.4 units before the rising edge of CLK2.

ptc_shell> set_input_delay 2.5 -clock CLK1 -clock_fall { INOUT1 }

ptc_shell> set_output_delay 1.4 -clock CLK2 { INOUT1 }

The following example models the situation where there are three paths
from output port OUT1. One of the paths is relative to the rising edge
of CLK1. Another path is relative to the falling edge of
CLK1. The third path is relative to the falling edge of
CLK2. The -add_delay option indicates that new output delay
information does not cause the removal of old information.

ptc_shell> set_output_delay 2.2 -max clock CLK1 -add_delay { OUT1 }

ptc_shell> set_output_delay 1.7 -max clock CLK1 -clock_fall -add_delay { OUT1 }

ptc_shell> set_output_delay 4.3 -max clock CLK2 -clock_fall -add_delay { OUT1 }

In the following example, two different maximum
delays and two minimum delays for
port Z are specified using -add_delay. Because
the information is relative
to the same clock and clock edge, only the largest of the maximum values
and the smallest of the minimum values are maintained (in this case,
5.0 and 1.1). If -add_delay is not used,
the new information overwrites the old information.

ptc_shell> set_output_delay 3.4 -max -clock CLK1 -add_delay { Z }

ptc_shell> set_output_delay 5.0 -max -clock CLK1 -add_delay { Z }
```

```
ptc_shell> set_output_delay 1.1 -min -clock CLK1 -add_delay { Z }

ptc_shell> set_output_delay 1.3 -min -clock CLK1 -add_delay { Z }

The following example shows how to use the -group_path option to add
ports into a named group. Note that without this option, paths to
these ports are included in the CLK group.

ptc_shell> set_output_delay 4.5 -max -clock CLK -group_path busA {busA[*]}
```

SEE ALSO

all_outputs(2)
create_clock(2)
current_design(2)
group_path(2)
remove_output_delay(2)
report_design(2)
report_port(2)
reset_design(2)
set_driving_cell(2)
set_load(2)
set_max_delay(2)
set_output_delay(2)

set_output_units

This command specifies the units that are used for output.

SYNTAX

```
string set_output_units
   [-time_unit time_unit ]
   [-resistance_unit resistance_unit ]
   [-capacitance_unit capacitance_unit ]
   [-voltage_unit voltage_unit ]
   [-current_unit current_unit ]
   [-power_unit power_unit ]

string  time_unit
string  resistance_unit
string  capacitance_unit
string  voltage_unit
string  current_unit
string  power_unit
```

ARGUMENTS

-time_unit *time_unit*

Specifies the time unit for user output. This can be of the form "1ns", "10.0ps", or "1.0e-12".

-resistance_unit *resistance_unit*

Specifies the resistance unit for user output. This can be of the form "1kOhm", "100.0Ohm", or "1.0e4".

-capacitance_unit *capacitance_unit*

Specifies the capacitance unit for user output. This can be of the form "1pF", "100.0fF", or "1.0e-12".

-voltage_unit *voltage_unit*

Specifies the voltage unit for user output. This can be of the form "1V", "100.0mV", or "1.0".

-current_unit *current_unit*

Specifies the current unit for user output. This can be of the form "1mA", "1000.0uA", or "1.0".

-power_unit *power_unit*

Specifies the power unit for user output. This can be of the form "1mW", "100.0mW", or "1.0".

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command specifies the units that are used for output such as reports. The tool maintains separate units for input and output. There are output unit values for time, resistance, capacitance, voltage, current, and power.

EXAMPLES

The following example sets the output units to 10ps, 1kOhm, 1pF, 1V, 1mA, and 1mW.

```
ptc_shell> set_output_units -time_unit 10ps -resistance_units 1kOhm \
 -capacitance_units 1pF -voltage_units 1V -current_units 1mA -power_units 1mW
```

SEE ALSO

get_output_unit(2)
get_input_unit(2)
set_input_units(2)

set_port_fanout_number

Sets the number of external fanout points on ports.

SYNTAX

```
string set_port_fanout_number
   [-min]
   [-max]
   fanout_number
   port_list
```

Data Types

```
   fanout_number      int
   port_list          list
```

ARGUMENTS

-min

Specifies the value for minimum condition.

-max

Specifies the value for maximum condition.

fanout_number

Specifies the number of external fanout points (Range: 0 to 100000).

port_list

Specifies a list of ports. Each element in the list is either a collection of ports or a pattern that matches ports on the current design.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the number of external fanout pins for ports in the current design.

To remove port **fanout_number** values, use the **remove_port_fanout_number** command.

EXAMPLES

This example specifies an external fanout number of 2 for port OUT1.

```
ptc_shell> set_port_fanout_number 2 [get_ports OUT1*]
```

SEE ALSO

remove_port_fanout_number(2)

set_propagated_clock

Specifies propagated clock latency.

SYNTAX

```
string set_propagated_clock object_list

list object_list
```

ARGUMENTS

object_list

Specifies a list of clocks, ports, or pins.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Specifies that delays be propagated through the clock network to determine latency at register clock pins. If not specified, ideal clocking is assumed. Ideal clocking means clock networks have a specified latency (designated by the **set_clock_latency** command), or zero latency, by default. Propagated clock latency is used for post-layout, after final clock tree generation. Ideal clock latency provides an estimate of the clock tree for pre-layout.

If the **set_propagated_clock** command is applied to pins or ports, it affects all register clock pins in the transitive fanout of the pins or ports. If it is applied to an object (clock, port, or pin) on which network latency is already defined, then a warning is issued.

Virtual clocks do not have any source, and they cannot affect any register in the design. Thus, virtual clocks cannot be made propagated, and an error is issued if set_propagated_clock is applied on a virtual clock.

To undo **set_propagated_clock**, use the **remove_propagated_clock** command or use the **set_clock_latency** command to provide an ideal latency.

To see propagated clock attributes on clocks, use the **report_clock** command.

Limitations: CRPR does not support propagated clocks set on pins or ports. If the variable **timing_remove_clock_reconvergence_pessimism** is set to TRUE then propagated clocks should be defined on clock objects, not pins or ports.

EXAMPLES

The following example specifies to use propagated clock latency for all clocks in the design.

```
ptc_shell> set_propagated_clock [all_clocks]
```

SEE ALSO

timing_remove_clock_reconvergence_pessimism(3)
remove_propagated_clock(2)
report_clock(2)
set_clock_latency(2)

set_rule_description

Changes the description of a rule.

SYNTAX

```
Boolean set_rule_description
   description_value
   rule

string description_value
string rule
```

ARGUMENTS

description_value

The new description text, to clarify the intent of the check. A good description will specify what conditions are checked and why this may be a problem.

rule

Rule name or collection of a single rule.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Modifies the description of specified rules. You can change the description of built-in or user-defined rules. The description is shown in the "Rule Definitions" output of **analyze_design**. You can get the description of a rule using **get_attribute $rule description**.

EXAMPLES

The following example changes the description for rule **UDEF_0009**.

```
ptc_shell> set_rule_description "Clocks should only be defined on input ports." UDEF_0009
```

SEE ALSO

analyze_design(2)
get_attribute(2)

set_rule_message

Changes the message list for a rule.

SYNTAX

```
Boolean set_rule_message
   message_list
   rule
```

Data Types

```
   message_list      list
   rule              string
```

ARGUMENTS

message_list

Specifies a list of message parts. Each part is a text string that is used together with parameter values to show each violation of this rule. The number of elements in this list must be the same as the number of elements for the existing message of this rule.

rule

Specifies the rule name or collection of a single rule.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Modifies the message of specified rules. You can change the message of built-in or user-defined rules. The message is shown in the output of the **analyze_design** command. You can get the message of a rule by setting **get_attribute $rule message**.

EXAMPLES

The following example changes the message for rule **UDEF_0009**.

```
ptc_shell> set_rule_message {"Clock " " is defined on an inout port source."} UDEF_0009
```

SEE ALSO

analyze_design(2)
get_attribute(2)

set_rule_property

Sets a property affecting how a specific rule check is performed, or how the output of a rule violation is presented.

SYNTAX

```
Boolean set_rule_property
   property_name
   property_value
   rule
```

Data Types

```
   property_name     string
   property_value    string
   rule              string
```

ARGUMENTS

property_name

Specifies the name of a valid property on the specified rule.

property_value

Specifies the new value for the property.

rule

Specifies the name of an existing rule. The named property is applied to this rule.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Modifies a property of one rule. Properties can control rule checking; for example, the limit of some

comparison. Properties can also control the output; for example, the maximum number of pins shown in the more info section of a violation.

EXAMPLES

The following example sets the **max_percent** property for rule **EXD_0009** to "50.0".

```
ptc_shell> set_rule_property max_percent 50.0 EXD_0009
```

SEE ALSO

analyze_design(2)
create_rule(2)
get_rule_property(2)

set_rule_severity

Sets the severity of a rule.

SYNTAX

```
Boolean set_rule_severity
   severity_value
   rule_list
```

Data Types

```
   severity_value    string
   patterns          list
```

ARGUMENTS

severity_value

Specifies the new severity value. Valid values are *info*, *warning*, *error*, or *fatal*.

rule_list

Lists the rule names or patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type rule.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Modifies the severity of specified rules. You can change the severity of built-in or user-defined rules. Severity is shown in the output of the **analyze_design** command. You can get the severity of a rule using **get_attribute $rule severity**.

EXAMPLES

The following example sets the severity for rule **EXD_0009** to "error".

```
ptc_shell> set_rule_severity error EXD_0009
```

SEE ALSO

analyze_design(2)
get_attribute(2)

set_sense

Specifies unateness propagating forward for pins with respect to clock source.

SYNTAX

```
string set_sense
   [-type type]
   [-positive]
   [-negative]
   [-stop_propagation]
   [-logical_stop_propagation]
   [-pulse pulse_type ]
   [-clocks clock_list]
   object_list
```

Data Types

```
   clock_list      list
   object_list     list
```

ARGUMENTS

-type *type*

Specifies whether the type of sense being applied refers to clock networks or data networks. The possible values for the *type* variable are: clock and data. Note that clock is assumed to be the default if the *-type* option is not given. Note that if *-type data* is given, the **-stop_propagation** and **-clocks** options must be used as well.

-positive

Specifies positive unateness applied to all pins in the *object_list* variable with respect to clock source. The *-positive* option cannot be specified with the *-negative* or *-pulse* options.

-negative

Specifies negative unateness applied to all pins in the *object_list* variable with respect to clock source. The *-positive* option cannot be specified with the *-negative* or *-pulse* options.

-stop_propagation

Stops the propagation of specified clocks in the *clock_list* variable from the specified pins or cell timing

arcs in the *object_list* variable. Only clock used as clock is stopped if used with *-type clock*. To stop propagation for both clock as clock and clock as data, you need two commands, one with *-type clock*, and one with *-type data*. The **-stop_propagation** option cannot be specified with the *-positive*, *-negative* or *-pulse* options.

-pulse *pulse_type*

Specifies the type of pulse clock applied to all pins in the *object_list* variable with respect to clock source. The possible values for the *pulse_type* variable are rise_triggered_high_pulse, fall_triggered_high_pulse, rise_triggered_low_pulse, or fall_triggered_low_pulse. The *-pulse* option cannot be specified with the *-negative* or *-pulse* options.

-clocks *clock_list*

Specifies a list of clock objects to be applied with the given unateness that is placed on all pin objects in the *object_list* variable. If the *-clocks* option is not supplied, all clocks passing through the given pin objects are considered.

object_list

Lists of pins or cell timing arcs with specified unateness to propagate. The timing arcs object can be used only with the *-stop_propagation* and *-logical_stop_propagation* options.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

This command gives you control to restrict unateness at pin (to positive or negative unate) with respect to clock source. However, the specified unateness only applies within the non-unate clock network. In this case, user-defined sense propagates forward from the given pins.

If the *-clocks* option is supplied, only the specified clock domains are applied. Otherwise, all clocks passing through the given pin objects are considered.

Constraint consistency issues warning messages if the specified sense on given pins cannot be respected in case there is predefined unateness for given pins. A hierarchical pin is not supported.

In addition, you can completely stop propagation of certain clocks in clock networks or data networks by using *-stop_propagation* option along with *-type* option.

To undo the **set_sense** command, use the **remove_sense** command.

EXAMPLES

The following example specifies a positive unateness for a pin named **XOR/Z** with respect to clock **CLK1**.

```
pt_shell> set_sense -positive -clocks [get_clocks CLK1] XOR/Z
```

The following example specifies negative unateness for a pin named **MUX/Z** for all clocks.

```
pt_shell> set_sense -negative MUX/Z
```

SEE ALSO

remove_sense(2)

set_size_only

Specifies that the given leaf cells should be sized only for optimization. This command applies only to ptc_shell

SYNTAX

```
string set_size_only
   [-all_instances]
   cell_list
   object_list
   [value]
```

Data Types

```
   cell_list     list
   object_list   list
   value         Boolean
```

ARGUMENTS

-all_instances

Specifies that a cell in the *object_list* is an instance whose parent cell is not unique. All similar instance leaf cells under the parent design should be set as size_only.

object_list

Specifies a list of cells, nets, designs, or library cells on which to place the touch.

value

Specifies the value with which to set. Allowed values are **true** (the default) or **false**.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets the given cells as size_only for optimization.

EXAMPLES

The following commands specify size_only on the block1 and analog1 cells, but not on net N1.

```
ptc_shell> set_dont_touch [get_cells {block1 analog1}]
1
ptc_shell> set_dont_touch [get_nets N1] false
1
```

SEE ALSO

set_dont_touch(2)

set_timing_derate

Sets derating factors for either the current design or a specified list of instances (cells, library cells, or nets).

SYNTAX

```
int set_timing_derate
   -early | -late
   [-rise] [-fall]
   [-clock] [-data]
   [-cell_delay] [-cell_check] [-net_delay]
   [-static] [-dynamic]
   [-scalar | -variation | -aocvm_guardband | -pocvm_guardband]
   [-pocvm_coefficient_scale_factor]
   [-increment]
   derate_value
   object_list
```

Data Types

```
   derate_value          float
   object_list           list
```

ARGUMENTS

-early

Indicates that the *derate_value* specified should be applied to early delays (shortest paths). This option and the *-late* option are mutually exclusive.

-late

Indicates that the *derate_value* specified should be applied to late delays (longest paths). This option and the *-early* option are mutually exclusive.

-rise

Indicates that the *derate_value* specified should be applied to rise delays.

-fall

Indicates that the *derate_value* specified should be applied to fall delays.

-clock

Indicates that the specified derating factor should apply only to clock paths.

-data

Indicates that the specified derating factors should apply only to data paths.

-net_delay

Indicates that the specified derating factor should apply to net delays. This option cannot be specified with the *-cell_check* option.

-cell_delay

Indicates that the specified derating factor should apply to cell delays. This option cannot be specified with the *-cell_check* option.

-cell_check

Indicates that the specified derating factor should apply only to cell timing checks. If this option is specified, the derate value set with the *-early* option is applied to hold and removal timing checks and the derate value set with the *-late* option is applied to setup and recovery timing checks. This option cannot be specified with any of the following options: *-clock*, *-data*, *-cell_delay*, *-net_delay*, *-aocvm_guardband*, and *-pocvm_guardband*.

-static

Indicates that the specified derating factor should apply to the nondelta portion of net delays only. This option requires that the *-net_delay* option is specified.

-dynamic

Indicates that the specified derating factor should apply to the dynamic component of delays only. This option requires that the *-net_delay* option is specified. This option cannot be specified with the *-aocvm_guardband* or *-pocvm_guardband* option.

-scalar

Indicates that the specified derating factor should apply to deterministic delays only.

-variation

Indicates that the specified derating factor should apply to statistical delays only.

-aocvm_guardband

This option is applicable only in an advanced on-chip variation (OCV) context. In an advanced OCV context, the derate factor that is applied to an arc is a product of the guard-band derate factor and the advanced OCV derate factor. This option cannot be specified with any of the following options: *-scalar*, *-variation*, *-dynamic*, or *-cell_check*.

-pocvm_guardband

This option is applicable only in a parametric on-chip variation (OCV) context. In a parametric OCV context, the derate factor that is applied to an arc is a product of the guard-band derate factor and the parametric OCV derate factor corresponding to a distance- based parametric OCV table, if specified. The parametric OCV factor applied to the standard deviation is also multiplied by the guardband factor. This option cannot be specified with any of the following options: *-scalar*, *-variation*, *-dynamic*, or -

cell_check.

-pocvm_coefficient_scale_factor

This option is applicable only in a parametric on-chip variation (OCV) context. In a parametric OCV context, the coefficient applied to an arc is a product of the parametric on-chip variation (POCV) coefficient and the parametric on-chip variation (POCV) coefficient scale factor.

-increment

Indicates that the specified derating factor is an incremental derate factor and should be added to the base derate factor to compute the total derate for an object. This option cannot be specified with the *-aocvm_guardband* option.

derate_value

Specifies the timing derate value that is applied to the specified delays as a scalar multiplicative factor.

object_list

Specifies the current design or a list of cells, library cells, or nets to which the specified derate factor is applied.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Sets derating factors for either the current design (if no object list is specified) or a set of cells, library cells, or nets in the current design.

Timing derate factors affect delay values shown in timing reports. Longest path delays (for example, launching paths for setup checks or capturing paths for hold checks) are multiplied by the derate value set using the *-late* option, and shortest paths (for example, capturing paths for setup checks or launching paths for hold checks) are multiplied by the derate values set using the *-early* option. If derate factors are not specified, then a value of 1.0 is assumed.

The intended usage mode of this command is that you first set global or default derates on the design and then you can use the *object_list* option to set specific derate values for cells, library cells, or nets in the design. To set derates globally on a design simply issue the command with no object list specified. For example, to set an early derate factor of value 0.95 on all cell and net delays in the design, use:

```
set_timing_derate -early 0.95
```

EXAMPLES

The following example sets an early (shortest path) derate factor of 0.7 and a late (longest path) derate factor of 1.0 globally on all cell and net delays the design:

```
ptc_shell> set_timing_derate -early 0.70
ptc_shell> set_timing_derate -late 1.0
```

The following example sets an early derate factor of 0.8 for the cell delay of cell U1:

```
ptc_shell> set_timing_derate -early 0.8 -cell_delay [get_cells U1]
```

The following example sets a late derate factor of 1.1 on all instances of the library cell IV in the library MY_LIB:

```
ptc_shell> set_timing_derate -late 1.1 [get_lib_cells MY_LIB/IV]
```

Assuming that cell "inv" is a particular instantiation of MY_LIB/IV in the design, the following command overwrites the late derate factor for the instance "inv" only:

```
ptc_shell> set_timing_derate -late 1.05 [get_cells inv]
```

The following example illustrates how to set a derate factor on all cells and nets (including subhierarchies) within the hierarchy top/H1:

```
ptc_shell> set_timing_derate -cell_delay -net_delay -late 1.05 [get_cells top/H1]
```

The following example shows how to set an early derate factor of 0.7 on a specific net 'n1':

```
ptc_shell> set_timing_derate -net_delay -early 0.7 [get_nets n1]
```

SEE ALSO

set_operating_conditions(2)

set_units

Checks the specified units with the main library units. The command fails if the units specified do not match the main library units.

SYNTAX

```
int set_units
   [-time time_in_sec]
   [-capacitance capacitance_in_farad]
   [-current current_in_ampere]
   [-voltage voltage_in_volt]
   [-resistance resistance_in_ohm]
   [-power power_in_watt]
```

Data Types

time_in_sec	float
capacitance_in_farad	float
current_in_ampere	float
voltage_in_volt	float
resistance_in_ohm	float
power_in_watt	float

ARGUMENTS

-time *time_in_sec*

Checks the time unit with the time unit of the main library.

-capacitance *capacitance_in_farad*

Checks the capacitance unit with the capacitance unit of the main library.

-current *current_in_ampere*

Checks the current unit with the current unit of the main library.

-voltage *voltage_in_volt*

Checks the voltage with the voltage unit of the main library.

-resistance *resistance_in_ohm*

Checks the resistance unit with the resistance unit of the main library.

-power ***power_in_watt***

Checks the power unit with the leakage power unit of the main library. The power units can be checked only if power analysis mode is enabled.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

The **set_units** command checks the units specified with the main library units. This command can be used only after the design has been linked. The main library units are the design units. The **set_units** command does not allow the setting of design units, but it is intended to prevent inadvertent use of the wrong units from different SDC files.

The **read_sdc** command supports the **set_units** command.

EXAMPLES

The following example checks the if the main library time unit is 1ns, capacitance unit is 1pF, current unit is 1mA, and voltage unit is 1V.

```
ptc_shell> set_units -time ns -capacitance pF -current mA -voltage V
```

The following example checks if the main library capacitance unit is 0.5pF.

```
ptc_shell> set_units -capacitance 0.5pF
```

SEE ALSO

read_sdc(2)

setenv

Sets the value of a system environment variable.

SYNTAX

```
string setenv variable_name new_value

string variable_name


string new_value
```

ARGUMENTS

variable_name

Names of the system environment variable to set.

new_value

Specifies the new value for the system environment variable.

DESCRIPTION

The **setenv** command sets the specified system environment *variable_name* to the *new_value* within the application. If the variable is not defined in the environment, the environment variable is created. The **setenv** command returns the new value of *variable_name*. To develop scripts that interact with the invoking shell, use **getenv** and **setenv**.

Environment variables are stored in the Tcl array variable **env**. The environment commands **getenv**, **setenv**, and **printenv** are convenience functions to interact with this array.

The **setenv** command sets the value of a variable only within the process of your current application. Child processes initiated from the application using the **exec** command after a usage of **setenv** inherit the new variable value. However, these new values are not exported to the parent process. Further, if you set

an environment variable using the appropriate system command in a shell you invoke using the **exec** command, that value is not reflected in the current application.

EXAMPLES

The following example changes the default printer.

```
shell> getenv PRINTER
laser1
shell> setenv PRINTER "laser3"
laser3
shell> getenv PRINTER
laser3
```

SEE ALSO

exec(2)
getenv(2)
unsettenv(2)
printenv(2)
printvar(2)
set(2)
sh(2)
unset(2)

sh

Executes a command in a child process.

SYNTAX

```
string sh [args]

string args
```

ARGUMENTS

args

Command and arguments that you want to execute in the child process.

DESCRIPTION

This is very similar to the **exec** command. However, file name expansion is performed on the arguments. Remember that quoting and grouping is in terms of Tcl. Arguments which contain spaces will need to be grouped with double quotes or curly braces. Tcl special characters which are being passed to system commands will need to be quoted and/or escaped for Tcl. See the examples below.

EXAMPLES

This example shows how you can remove files with a wildcard.

```
prompt> ls aaa*
aaa1      aaa2      aaa3
prompt> sh rm aaa*
prompt> ls aaa*
```

```
Error: aaa*: No such file or directory
       Use error_info for more info. (CMD-013)
```

This example shows how to grep some files for a regular expression which contains spaces and Tcl special characters:

```
prompt> exec cat test3.out
blah blah blah
blah blah blah c blah
input [1:0] A;
output [2:0] B;
prompt>
prompt> sh egrep -v {[ ]+c[ ]+} test3.out
blah blah blah
prompt>
prompt> sh egrep {t[ ]+\[} test3.out
input [1:0] A;
output [2:0] B;
```

SEE ALSO

exec(2)

sizeof_collection

Returns the number of objects in a collection.

SYNTAX

```
int sizeof_collection
   collection1
```

Data Types

```
   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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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.

```
ptc_shell> 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.

```
ptc_shell> set s1 [get_cells *]
{"u1", "u2", "u3"}
ptc_shell> set ssize [filter_collection $s1 "area < 0"]
ptc_shell> echo "Cells with area < 0: [sizeof_collection $ssize]"
Cells with area < 0: 0
ptc_shell> 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]
   [-dictionary]
   collection1
   criteria
```

Data Types

```
   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.

-dictionary

Sorts the strings in dictionary order. For example "a30" would come after "a4".

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

This command is available only if you invoke the pt_shell with the **-constraints** option.

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.

```
ptc_shell> set zcells [get_cells {o2 i2 o1 i1}]
{"o1", "i2", "o1", "i1"}
ptc_shell> set zsort [sort_collection $zc {is_hierarchical full_name}]
{"o1", "o2", "i1", "i2"}
```

SEE ALSO

collections(2)

source

Read a file and evaluate it as a Tcl script.

SYNTAX

```
string source [-echo] [-verbose] [-continue_on_error] file

string file
```

ARGUMENTS

-echo

Echoes each command as it is executed. Note that this option is a non-standard extension to Tcl.

-verbose

Displays the result of each command executed. Note that error messages are displayed regardless. Also note that this option is a non-standard extension to Tcl.

-continue_on_error

Don't stop script on errors. Similar to setting the shell variable sh_continue_on_error to true, but only applies to this particular script.

file

Script file to read.

DESCRIPTION

The **source** command takes the contents of the specified *file* and passes it to the command interpreter as a text script. The result of the source command is the result of the last command executed from the file. If an error occurs in evaluating the contents of the script, then the **source** command returns that error. If a return command is invoked from within the file, the remainder of the file is skipped and the source command returns normally with the result from the return command.

By default, source works quietly, like UNIX. It is possible to get various other intermediate information from the source command using the **-echo** and **-verbose** options. The **-echo** option echoes each command as it appears in the script. The **-verbose** option echoes the result of each command after execution.

NOTE: To emulate the behavior of the dc_shell **include** command, use both of these options.

The file name can be a fully expanded file name and can begin with a tilde. Under normal circumstances, the file is searched for based only on what you typed. However, if the system variable sh_source_uses_search_path is set to "true", the file is searched for based on the path established with the search_path variable.

The **source** command supports several file formats. The *file* can be a simple ascii script file, an ascii script file compressed with gzip, or a Tcl bytecode script, created by TclPro Compiler. The **-echo** and **-verbose** options are ignored for gzip formatted files.

EXAMPLES

This example reads in a script of aliases:

```
prompt> source -echo aliases.tcl
alias q quit
alias hv {help -verbose}
alias include {source -echo -verbose}
prompt>
```

SEE ALSO

search_path(3)
sh_source_uses_search_path(3)
sh_continue_on_error(3)

suppress_message

Disables printing of one or more informational or warning messages.

SYNTAX

```
string suppress_message [message_list]

list message_list
```

ARGUMENTS

message_list

A list of messages to suppress.

DESCRIPTION

The **suppress_message** command provides a mechanism to disable the printing of messages. You can suppress only informational and warning messages. The result of **suppress_message** is always the empty string.

A given message can be suppressed more than once. So, a message must be unsuppressed (using **unsuppress_message**) as many times as it was suppressed in order for it to be enabled. The **print_suppressed_messages** command displays the currently suppressed messages.

EXAMPLES

When the argument to the **unalias** command does not match any existing aliases, the CMD-029 warning message displays. This example shows how to suppress the CMD-029 message:

```
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)
get_message_ids(2)
set_message_info(2)

unalias

Removes one or more aliases.

SYNTAX

```
string unalias
   patterns
```

ARGUMENTS

patterns

Specifies the patterns to be matched. This argument can contain more than one pattern. Each pattern can be the name of a specific alias to be removed or a pattern containing the wildcard characters ***** and **%**, which match one or more aliases to be removed.

DESCRIPTION

The **unalias** command removes aliases created by the **alias** command.

EXAMPLES

The following command removes all aliases.

```
prompt> unalias *
```

The following command removes all aliases beginning with f, and the alias rt100.

```
prompt> unalias f* rt100
```

SEE ALSO

alias(2)

unsetenv

Removes a system environment variable.

SYNTAX

```
string getenv
   variable_name
```

Data Types

```
   variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of the environment variable to be unset.

DESCRIPTION

The **unsetenv** command searches the system environment for the specified *variable_name* and removes variable from the environment. If the variable is not defined in the environment, the command returns a Tcl error. The command is catchable.

Environment variables are stored in the **env** Tcl array variable. The **unsetenv**, commands is a convenience function to interact with this array. It is equivalent to 'unset ::env(*variable_name*)'

The application you are running inherited the initial values for environment variables from its parent process (that is, the shell from which you invoked the application). If you unset the variable using the **unsetenv** command, you remove the variable value in the application and in any new child processes you initiate from the application using the **exec** command. However, the variable is still set in the parent process.

See the **set** and **unset** commands for information about working with non-environment variables.

EXAMPLES

In the following example, **unsetenv** remove the DISPLAY varible from the environment:

```
prompt> getenv DISPLAY
host:0
prompt> unsetenv DISPLAY
prompt> getenv DISPLAY
Error: can't read "::env(DISPLAY)": no such variable
        Use error_info for more info. (CMD-013)
```

SEE ALSO

```
catch(2)
exec(2)
printenv(2)
set(2)
unset(2)
setenv(2)
getenv(2)
```

unsuppress_message

Enables printing of one or more suppressed informational or suppressed warning messages.

SYNTAX

```
string unsuppress_message [messages]

list messages
```

ARGUMENTS

messages

A list of messages to enable.

DESCRIPTION

The **unsuppress_message** command provides a mechanism to re-enable the printing of messages which have been suppressed using **suppress_message**. You can suppress only informational and warning messages, so the **unsuppress_message** command is only useful for informational and warning messages. The result of **unsuppress_message** is always the empty string.

You can suppress a given message more than once. So, you must unsuppress a message as many times as it was suppressed in order to enable it. The **print_suppressed_messages** command displays currently suppressed messages.

EXAMPLES

When the argument to the **unalias** command does not match any existing aliases, the CMD-029 warning message displays. This example shows how to re-enable the suppressed CMD-029 message. Assume 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)
```

SEE ALSO

print_suppressed_messages(2)
suppress_message(2)

which

Locates a file and displays its pathname.

SYNTAX

```
string which filename_list

list filename_list
```

ARGUMENTS

filename_list

List of files to locate.

DESCRIPTION

Displays the location of the specified files. This command uses the search_path to find the location of the files. This command can be a useful prelude to read_db or link_design, because it shows how these commands expand filenames. The **which** command can be used to verify that a file exists in the system.

If an absolute pathname is given, the command searches for the file in the given path and returns the full pathname of the file.

EXAMPLES

The following examples are based on the following search_path.

prompt> **set search_path "/u/foo /u/foo/test"**

The following command searches for the file name foo1 in the search_path.

```
prompt> which foo1
/u/foo/foo1
```

The following command searches for files foo2, foo3.

```
prompt> which {foo2 foo3}
/u/foo/test/foo2 /u/foo/test/foo3
```

The following command returns the full pathname.

```
prompt> which ~/test/designs/sub_design.db
/u/foo/test/designs/sub_design.db
```

SEE ALSO

link_design(2)
read_db(2)
search_path(3)

write_app_var

Writes a script to set the current variable values.

SYNTAX

```
string write_app_var
   -output file
   [-all | -only_changed_vars]
   [pattern]
```

Data Types

```
   file         string
   pattern      string
```

ARGUMENTS

-output *file*

Specifies the file to which to write the script.

-all

Writes the default values in addition to the current values of the variables.

-only_changed_vars

Writes only the changed variables. This is the default when no options are specified.

pattern

Writes the variables that match the specified *pattern*. The default is "*".

DESCRIPTION

The **write_app_var** command generates a Tcl script to set all application variables to their current

values. By default, variables set to their default values are not included in the script. You can force the default values to be included by specifying the **-all** option.

EXAMPLES

The following is an example of the **write_app_var** command:

```
prompt> write_app_var -output sh_settings.tcl sh*
```

SEE ALSO

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

write_waiver

Writes existing waivers into an output file. The file can be sourced to restore waivers in a new session.

SYNTAX

```
Boolean write_waiver
   -output output_file
   [-force]
```

Data Types

```
   output_file     string
```

ARGUMENTS

-output *output_file*

Writes all existing waivers into the given output file.

-force

Overwrites the output file if it already exists.

DESCRIPTION

This command is available only if you invoke the pt_shell with the **-constraints** option.

Writes existing waivers into an output waiver file. To restore waivers from the **write_waiver** output, you need to source the waiver file in each scenario. Sourcing a waiver file in one scenario restores only the waivers active in that scenario, and waivers for scenario-independent rules.

SEE ALSO

```
analyze_design(2)
create_waiver(2)
report_waiver(2)
remove_waiver(2)
```