SPECCTRA tools
Autorouting Commands

Product Version 9.0
September 1999

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Table of Contents

Overview

This book contains autorouting command reference information that is derived from the SPECCTRA 9.0 online help. The original design intent and organization of information in this document is for online access. For more complete information, refer to the SPECCTRA online help (by clicking on the Help menu). The SPECCTRA online help includes placement and other command references not included in this book and user interface information, procedures, and general purpose information. Online books are also accessible from the Help menu.

About SPECCTRA Commands

The autorouting command reference provides syntax diagrams, general descriptions, and examples for SPECCTRA commands.

See Syntax Conventions in this section for an introduction to the conventions used in the syntax diagrams.

To use a command in SPECCTRA, you can

- Type the command in the command entry area
- Include the command in a do file

You can enter multiple commands on the same line by separating them with semicolons. SPECCTRA performs the commands sequentially. For example, to route 25 routing passes and 2 clean passes, enter

```
route 25;clean 2
```

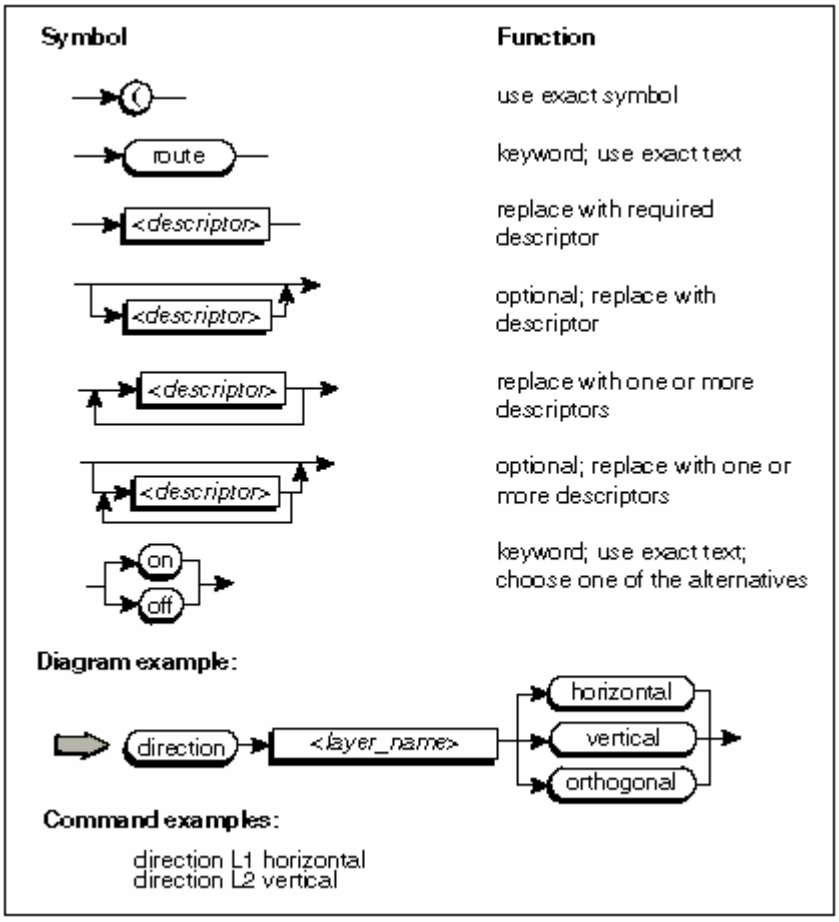
You can also use SPECCTRA menus and dialog boxes to apply commands. The SPECCTRA online help provides information about using menus and dialog boxes.

If you set or modify rules during a SPECCTRA session, and want to use them in another session, you can modify the did file and use it as a do file. After the session, open the did file in a text editor, and remove any command you do not want to use in the do file.

If you are a new SPECCTRA user, you should also read about design object names, file naming conventions, and the SPECCTRA design rule hierarchy in the SPECCTRA online help.

Command Syntax Conventions

The following symbols and example explain the symbols used to diagram SPECCTRA syntax.



Conventions Used in This Manual

The following fonts, characters, and styles have specific meanings throughout this manual.

- **Boldface** type identifies text that you type exactly as shown, such as SPECCTRA command names, keywords, and other syntax elements. In the following example, **average_pair_length**, **on**, and **off** are keywords.

  (**average_pair_length** [**on** | **off**])

  Syntax or command examples that appear on a separate line are not bold.

  (boundary (rect pcb 0 0 9000 4000))

  For information about the Backus-Naur Form (BNF) metalanguage conventions used to represent the design language, see *SPECCTRA Design Language Reference*.

- *Italic* type identifies titles of books and emphasizes portions of text.

  See the *SPECCTRA Installation and Configuration Guide* for information about installing SPECCTRA.

Italicized words enclosed in angle brackets (<>) are placeholders for keywords, values, filenames, or other information that you must supply.

<*directory_path_name*>::= <*id*>

- References to keys on your keyboard and mouse buttons are enclosed in brackets. [Shift] refers to the shift key. The carriage return key is labeled "Enter" on some keyboards and "Return" on others. This manual uses [Enter].

The following special terms are used in this manual.

- The word *enter* used with commands means type the command and press [Enter].

  “Enter the command **grid wire 1**” means

  1. Type **grid wire 1**.
  2. Press [Enter].

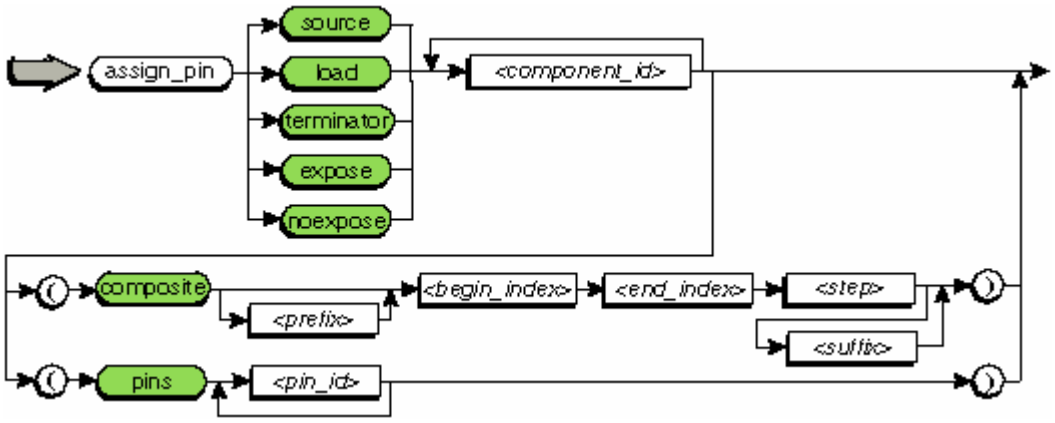
- *Click* means press and release the left mouse button.
- *Click-middle* means press and release the middle mouse button.
- *Click-right* means press and release the right mouse button.
- *Drag* means press and hold the left mouse button while you move the pointer.
- *Drag* [MB] means press and hold the middle mouse button while you move the pointer.
- *Double-click* means press and release the left mouse button twice in rapid succession.
- *Click twice* means click twice at the same location in the SPECCTRA work area.
- *Select* means to identity objects (such as wires, nets, or components) for exclusive processing by routing or placement commands. When you *select* objects before using a command, the autorouter works only on the objects that are selected.
- *Switch* refers to one or more characters you can use with an operating system command, such as the command you use to start SPECCTRA. A hyphen (-) precedes each command line switch.

Autorouting Command Reference

This section provides syntax drawings, general descriptions, and examples of SPECCTRA autorouting commands. The commands appear in alphabetical order.

assign_pin

The **assign_pin** command assigns source, load, terminator, and expose properties to component pins.



source

A property assigned to pins for daisy-chain routing.

load

A property assigned to pins for daisy-chain routing. All pins default to load unless the source or terminator property is assigned.

terminator

A property assigned to pins for daisy-chain routing.

expose

An attribute that forces a through-pin escape to a via on an external PCB layer. The **expose** attribute applies to through-pins only.

noexpose

An attribute that removes the **expose** attribute for the specified pins so that fanout does not generate vias for those pins.

composite

Identifies a list of pin names that match the <*begin_index*>, <*end_index*>, <*step*>, and optional <*prefix*> and <*suffix*> parameters.

<prefix>

A non-numeric character or character string that represents a pin number prefix. For example, A or 1A.

<begin_index>

Pin number of the first pin in the list.

<end_index>

Pin number of the last pin in the list.

<step>

Increment that determines pins included between the start index pin and the end index pin.

<suffix>

A single character or character string that represents a pin number suffix. For example, B or BB.

pins

Identifies a list of pins by <*pin_id*>. The <pin_id> must not contain a hyphen.

The **assign_pin** command provides an efficient way to specify large numbers of daisy-chained nets in source-load-terminator format.

When you assign the **expose** property to a through-pin, the pin escapes to a via on an external PCB layer. If the autorouter needs to connect on an internal layer, it routes to the via.

By specifying **direction** with the **fanout** command you can control whether through-pins with the **expose** property escape outside the component outline. You can direct the autorouter to escape wires and vias inside the component outline (**in**), outside (**out**), or both (**in_out**). When the **in_out** option is set for **fanout** (default), exposed pins escape outside the component outline. See the fanout command.

The **assign_pin** command is particularly effective when several nets start as sources on multiple components and terminate on another set of components.

The following describes a strategy for using **assign_pin**.

To specify a large number of daisy-chained nets in source-load-terminator format.

1. Change the property to source for a large number of pins on one or more components by using the **assign_pin source** command. For example:

assign_pin source U27 U28 U29

2. Change the property to terminator for a second group of pins by using the **assign_pin terminator** command. For example:

assign_pin terminator RN27 RN28 RN29

Pins that are not assigned source or terminator properties default to load.

3. Reorder the affected pins for daisy-chain routing by using the order command. For example:

order daisy net sig1 sig2 sig3

You can also use the define class command to create a class of nets and order them as daisy-chain. When you order a net for daisy-chain routing, multiple source and terminator pins are optimally chained.

4. Reset the components that were not ordered as daisy-chain to the load property by using the **assign_pin load** command. For example:

assign_pin load U27 U28 U29

See also the define net command which can be used to control the exact pin order for a net.

Command examples

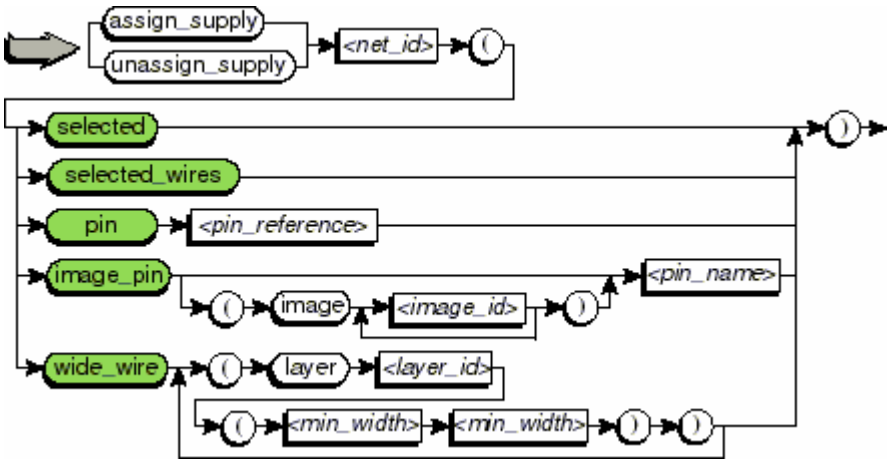
assign_pin source U200 (pins 2 3 5 7 8)
assign_pin load U201 (composite A 2 20 2)
assign_pin terminator R???

assign_pin source U200
assign_pin terminator U400
order daisy net clock
assign_pin load U200 U400

assign_pin expose U202
assign_pin noexpose U202 (pins 4 6)

assign_supply

The **assign_supply** command identifies the component pins or wires of a power net as a supply trunk. The **unassign_supply** command returns component pins or wires to normal status.



selected

Use this option to create a trunk that includes all currently selected wires and component pins.

selected_wire

Use this option to create a trunk that includes only currently selected wires.

pin

Use this option to create a trunk that includes a component pin that you specify. The <*pin_reference*> consists of a component name, a hyphen, and a pin name.

image_pin

Use this option to assign all pins on the net with the specified <*pin_name*> to the supply trunk. Use the **image** option to assign only pins on the specified <*image_id*>.

wide_wire

Use this option to assign existing wires in the net to the supply trunk if the wire width is at least <*min_width*>.

This command identifies certain component pins or selected wires that must be routed directly to the power source. You identify the name (<net_id>) of the power net and the pins or wires that constitute the supply trunk. A trunk can consist of one or more specific component pins, selected pins and wires, or just selected wires.

- Use **selected** to create the trunk with all currently selected component pins and wires. You must select the pins or wires before using **assign_supply**.
- Use **selected wires** to create the trunk with all currently selected wires. You must select the wires before using **assign_supply**.
- Use **pin** to create the trunk with a specific component pin. The <*pin_reference*> consists of a component name, a hyphen, and a pin number.
- Use **image_pin** to assign specified image pins to the supply trunk.

- Use **wide_wire** to assign existing wires in the net to the supply trunk if the wires are equal to or greater than the specified width.

Note

The pins or wires need not be interconnected, but the autorouter must connect other pins on the net to a point on the supply pin or trunk.

Use the **image_pin** keyword without the **image** option to assign all pins named <*pin_name*> in the specified net.

You can also use this command to treat pins and wires of any net as a trunk.

See also

junction_type rule to control routing topology for any pins and wires defined as a trunk with the **assign_supply** command.

Examples

assign_supply vcc (pin C1-A)
unassign_supply vcc (pin C1-A)

assign_supply vcc (selected)
unassign_supply vcc (selected)

assign_supply vcc (selected_wires)
unassign_supply vcc (selected_wires)

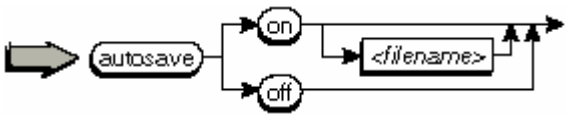
assign_supply vcc (image_pin vcc)
unassign_supply vcc (image_pin vcc)

assign_supply vcc (wide_wire (layer M1 (min_width 10)))
unassign_supply vcc (wide_wire (layer M1 (min_width 10)))

assign_supply vcc (wide_wire (layer M1 (min_width 10)) (layer M2 (min_width 20)))
unassign_supply vcc (wide_wire (layer M1 (min_width 10)) (layer M2 (min_width 20)))

autosave

The **autosave** command controls whether wires are saved after each routing pass.



This command turns the **autosave** function on and off. If the function is turned **on**, the autorouter writes the wiring results to a file at the end of each routing pass. This is an overwrite process. At the end of an autorouting session, or in the event of a system crash, the results of the most recent wiring pass are in the autosave file. You can use the autosave file to recover. The default filename is autosave.w.

Use the write wire command instead of **autosave** to save wires at the end of a session. This protects your final wires file from an accidental overwrite during a subsequent autorouting session.

See also the bestsave command, which is preferred over **autosave**.

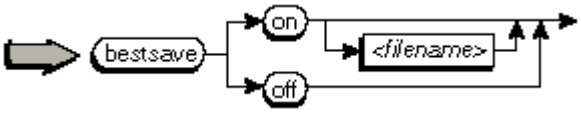
Command examples

autosave on
autosave on mysave.w

For general information about specifying filenames, see File Naming Conventions.

bestsave

The **bestsave** command controls whether wires are saved when there is a routing improvement.



This command turns the **bestsave** function on and off. When **bestsave** is **on**, the autorouter writes the routed wires at the end of each routing pass if the wiring has improved since the previous **bestsave**.

SPECCTRA calculates a routing pass-score as follows:

```
pass-score = crossing violations
           + clearance violations
           + crosstalk violations
           + length violations
           + 2 * unroutes
```

Use the wires file that is created by **bestsave** to recover your work in the event of a power failure. The default filename is bestsave.w.

The **bestsave** command does not replace the **write wire** command, which is used at the conclusion of an autorouting session. See the write wire command.

You load the wires to restart an autorouting session by using the read wire command.

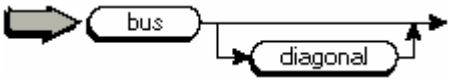
Command examples

bestsave on
bestsave on mysave.w

For general information about specifying filenames, see File Naming Conventions.

bus

The **bus** command uses a special algorithm to route component pins that share the same X or Y coordinates.

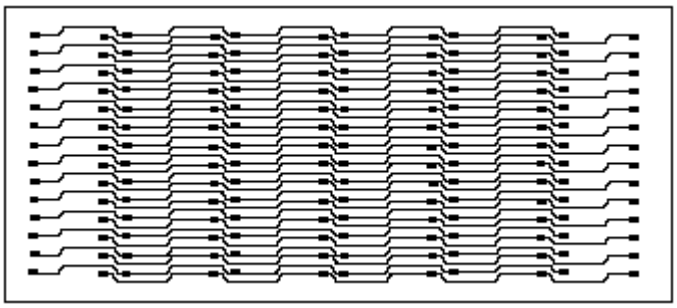


The **bus** command directs the autorouter to route regular arrays of pins such as those that interconnect memory devices. The autorouter determines which nets are

candidates for bus routing and then routes these connections. Clearance rules must permit sufficient space to allow bus routing without conflicts.

If you use the **bus** command without the **diagonal** option, buses are routed orthogonally. Bus-diagonal routing is preferred because it provides the highest routing density.

An example of **bus diagonal** routing is shown in the following illustration.

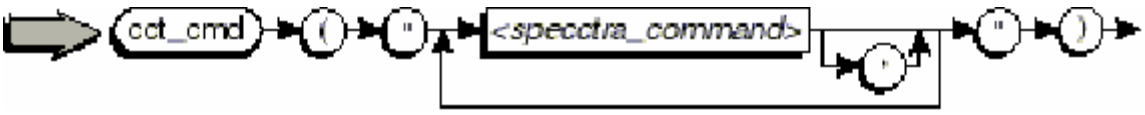


Command examples

bus
bus diagonal

cct_cmd

The **cct_cmd** command allows you to issue SPECCTRA commands while the command entry area is set to SKILL mode (see skill_mode).



Thus, the **cct_cmd** command is useful when executing SKILL do files. To enter multiple SPECCTRAcommands, separate the commands with a semicolon (;).

Command examples

skill_mode
printf(“total components = %d\n” totalcomp)
for (i 0 5{cct_cmd(“z out”)})
cct_mode

skill_mode
cct_cmd(“z out 2; repaint”)

See also

skill_cmd
skill_mode
cct_mode

cct_mode

The **cct_mode** command sets the SPECCTRA command entry area to accept SPECCTRA commands. You typically use this command to exit SKILL mode (see skill_mode).

Command example

```
skill_mode
printf("total components = %d\n" totalcomp)
cct_mode
```

See also

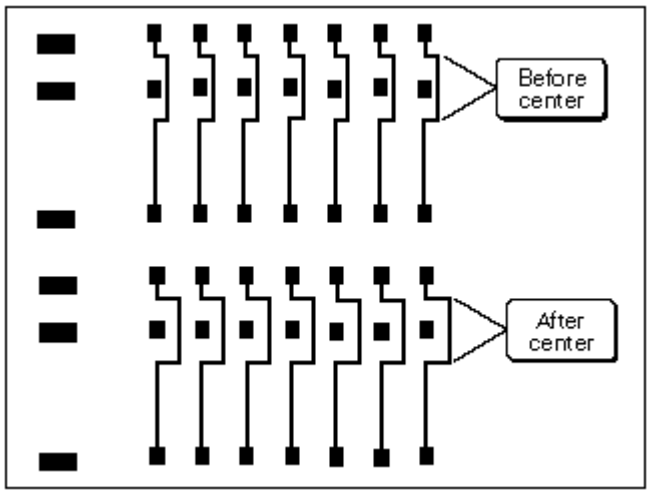
skill_cmd
skill_mode
cct_cmd

center

The **center** command attempts to move single wire segments so that they are equidistant between adjacent pins of a component.

The **center** command examines all wires that pass between adjacent pins of a component and positions these wire segments equidistant between the pins, subject to the following conditions:

- No new conflicts are introduced.
- Only a single wire segment lies between a pin pair (per layer).
- No new routing segments are required to achieve centering. Only a single segment move is permitted.
- No additional bends are added to wires.
- If a wiring grid is defined, wires are placed on the grid closest to the center line between the pins.



See also

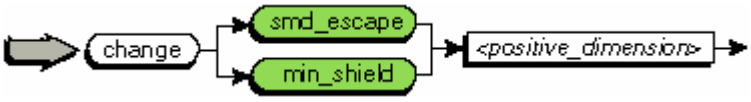
spread

Command example

center

change

The **change** command controls fanout escape distance for SMDs on an unselected layer, which is the maximum wire length from the pin that the autorouter can place a via, and the minimum shielded segment length.



smd_escape

When you unselect the layers that SMDs are mounted on, the autorouter must still route to the escape vias. The default maximum escape distance used by the autorouter is 0.25 inches from the edge of the SMD pad to the center of the via.

Escape wires and vias can be rerouted throughout the autorouting session as part of the normal rip-up and reroute process. The escape vias are always positioned within the <*smd_escape*> radius.

min_shield

This parameter specifies the minimum terminal-to-terminal connection length the autorouter attempts to enclose in a shield wire when the circuit shield command is used. The default min_shield value is 0.125 inches. This distance is measured from terminal-origin to terminal-origin.

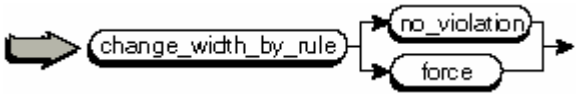
The **change** command controls the maximum wire length that is used to escape SMD pads on an unselected and the minimum segment length that is shielded.

Command examples

unit inch
change smd_escape 500
change min_shield 0.5

change_width_by_rule

The change_width_by_rule command changes the width of wires affected by a width rule change.



Use this command to automatically update the width of all wires affected by a change in wire width rules. By default, changes to wire width are made only where violations are not created. Use the **force** option to make all width changes based on the new rules, even if the change creates a violation.

Note

This command does not apply to wires affected by changes in region rules.

See also

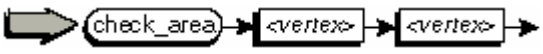
highlight

Command examples

rule layer s2 (width 1)
rule layer s3 (width 2)
change_width_by_rule force

check_area

The **check_area** command examines the design within a rectangular area to determine both placement and routing design rule violations.



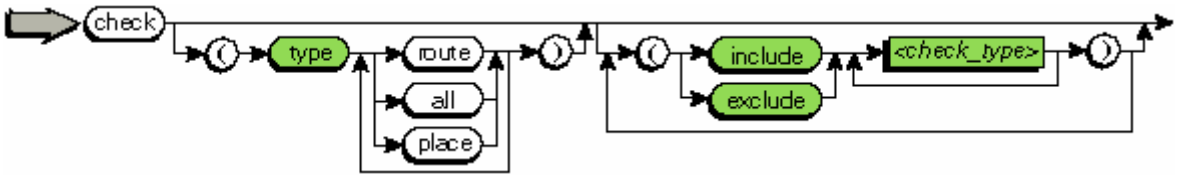
The command marks violations that exist within the specified area. Use the command to evaluate the effects of rule changes within an area, and to detect design rule errors created when checking is off, without having to check the entire design. Use the check command to list all violations in the design.

Command example

check_area 1.1 0.2 2.0 0.5

check

The **check** command examines the design to determine placement and routing rule violations.



type

Controls which rules are checked. The choices are

route, which means check routing rules. Clearance violations are marked with rectangles, crossover violations with diamonds, length violations with dashed lines, and crosstalk violations with thin-lined rectangles.

all, which means check routing rules, placement rules, and all other options available for **type**.

place, which means check placement rules. Violations are marked with a thick-lined rectangle and with small diamonds on each corner of the component outline.

include

Adds check_type options to the list of rules and objects the routing checker checks in this command execution only.

exclude

Removes check_type options from the list of rules and objects the routing checker checks in this command execution only.

<check_type>

The check types you can specify with the setup_check and check commands are:

Type	Defaults to
conflict	on
length	on
limit_way	off
max_vias	off
miter	off
order	off
pin	off
polygon_wire	off
protected	off
same_net_check	off
stagger	off
stub	off

use_layer	off
use_via	off
xtalk (crosstalk)	on

Use the **check** command to evaluate the effects of rule changes or to find placement and routing rule violations that occur during interactive operations while rule checking is turned off. You can use the **type** option to check:

place - just placement rules, and comp_outline if set in the setup_check command, or

route - all the routing checker options specified in the setup_check command, or

all - all the options available for **type**

Use the **type** keyword to identify the type of checking you want to perform (place, route, or all).

You can use the **include** and **exclude** keywords to add one or more checking options that are turned off in the current setup, or to remove checking options that are currently on. The **include** and **exclude** keywords do not change the checking setup, they only apply to the check command with which they are used.

If you use **check** without the **type, include**, or **exclude** keywords, only routing violations are checked. This is equivalent to **check (type route)**.

SPECCTRA automatically checks for rule violations at the beginning of a session and after every placement or routing operation. If you add or change a rule during a session, you can use the **check** command to evaluate the effects of the new rule.

If you turn off checking, modify the routing using interactive tools, and then turn on rule checking, SPECCTRA does not immediately check for rule violations. You must issue the **check** command to find rule violations that occurred when rule checking was turned off.

Placement checking works differently. If checking is turned on, you can't create a placement violation. If checking is turned off when you move a component, and you create a placement violation, the violation is marked immediately. Note that placement violations are displayed graphically only if the Placement Errors layer is displayed in the Layers panel.

See also

setup_check
place_rule
circuit
rule

Command examples

check

check (type route)

check (type place)

check (type all)

check (include miter stub limit_way) (exclude xtalk)

<check_type>

conflict

Checks for shorts and clearance violations.

The default is **on**.

length

Checks for violations of length rules.

The default is **on**.

limit_way

Checks for violations of the **rule** command limit_way rule.

The default is **off**.

max_vias

Controls whether the maximum via rules for nets, classes, groups, and fromtos are checked. The default setting for this control is **off**, which means maximum via rules are not checked. See the **rule** command for setting max_vias rules.

miter

Checks for unmitered wire corners.

The default is **off**.

order

Checks routed wiring for violations of the net ordering rules, and highlights violations in the work area when you run the **check** command.

Note

You might not want to turn on both **order** and **stub** at the same time because the violations appear similar when highlighted in the work area.

pin

Checks for clearance violations between pins and other objects.

The default is **off**.

polygon_wire

Checks for clearance violations between wiring polygons and other objects.

The default is **off**.

protected

Checks for clearance violations between protected wires or vias and other objects.

The default is **off**.

same_net_check

Checks for clearance rule violations between objects on the same net. A same net clearance rule violation occurs when a wire segment, via, or pin is too close to another object on the same net.

The default is **off**.

Note

The via_via and via_via_same_net clearance rules are always checked and are not affected by this control. Only clearance rules, which are used to prevent unintended shorts, are checked.

stagger

Checks for violations of the **rule** command maximum stagger rule.

The default is **off**.

stub

Checks for violations of the **rule** command max_stub length rule, and highlights violations in the work area when you run the **check** command.

The default is **off**.

Note

You might not want to turn on both **stub** and **order** at the same time because the violations appear similar when highlighted in the work area.

use_layer

Checks for violations of the **circuit** command use_layer rule.

The default is **off**.

use_via

Checks for violations of the **circuit** command use_via rule.

The default is **off**.

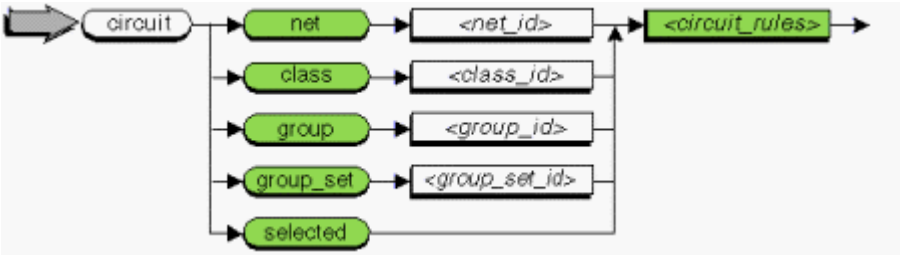
xtalk

Checks for violations of crosstalk rules.

The default is **on**.

circuit

The **circuit** command assigns rules to nets, net classes, fromtos, groups of fromtos, and group sets.



net

Applies circuit rules to the specified <*net_id*>. The <*net_id*> is the name of a net defined in SPECCTRA or in the design file.

class

Applies circuit rules to the specified <*class_id*>. The <*class_id*> is the name of a class defined in SPECCTRA or in the design file.

group

Applies circuit rules to the specified <*group_id*>. The <*group_id*> is the name of a group defined in SPECCTRA or in the design file.

group_set

Applies circuit rules to the specified <*group_set_id*>. The <*group_set_id*> is the name of a group set defined in SPECCTRA or in the design file.

selected

Applies circuit rules to only the selected nets.

Use the **circuit** command to assign length, delay, and shielding rules, and routing priorities, vias, and routing layers. See circuit rules overview for general information about circuit rules. Some **circuit** rules do not apply to all objects.

The object keyword you use determines at which level of rule precedence you want SPECCTRA to apply your routing rules. The choices are **net, class**, **group**, **and group_set**. For a list of the general types of rules that apply to each rule precedence level see routing rule hierarchy. You can also use the **selected** keyword to apply rules to selected nets. Use <*circuit_rules*> to set your rules.

The objects you can apply **circuit** rules to are described below.

Object	Description
net	A single net name.

class	One or more nets that share common rules. Use the define class command to create a unique class name and assign nets.
group	A set of fromtos that share common rules. Use the define group command to create a unique group name and assign fromtos.
group_set	A set of groups that share common rules. Use the define group_set command to create a unique group set name.
selected	One or more nets that are marked by using select net. A complete net must be selected. This command does not work on selected fromtos.

Note: Use the define command to assign rules to fromtos at the net, group, or group set levels.

You can control maximum and minimum routed lengths, match the routed lengths of two or more nets, match the routed lengths of the fromtos in a net, control the total length of a group of fromtos, match the routed lengths of groups in a group set, and automatically route shields for nets. You can specify length rules by using:

- Actual dimensions
- Ratio of routed length versus Manhattan length
- Delay values in units of time

If you use delay rules, you must define a time_length_factor by using the rule command. A warning message appears if you try to set a delay rule without setting a time_length_factor.

Note

You cannot set length rules using both actual dimensions and time units. If you set a length rule by using delay, all prior length rules that use actual dimensions are ignored. If you set a length rule by using actual dimensions, all prior length rules that use delay are ignored.

Command examples

```
unit mil
circuit net GND (use_via V100)
define (class c1 sig1 sig2 sig3 (circuit (priority 255)))
circuit class c1 (match_net_length on (ratio_tolerance 20))
circuit selected (use_layer L2 L3)

define (group g1 (fromto U1-3 U3-4)
   (fromto U6-8 U7-5)
   (circuit (length 3000 2400)))
circuit group g1 (match_fromto_length on (tolerance 0.10))
circuit net J1 (length 4.4 3.9 (type ratio))
```

```
circuit net AR0 (length -1 2.5)
rule net J1 (length_gap 0.008)
rule net AR0 (length_amplitude 0.25)

define (class c2 sig1 sig2 sig3 (circuit (max_delay 525)
   (min_delay 420)))
rule class c2 (time_length_factor .45)

define (group g2 (fromto U4-20 U2-17) (fromto U4-2 U4-8))
define (group g3 (fromto U1-6 U3-12) (fromto U1-16 U3-7))
define (group g4 (fromto U2-9 U3-20) (fromto U2-7 U3-8))
define (group_set grpset1 g2 g3 g4)
circuit (group_set grpset1 (match_group_length on
   (ratio_tolerance 15))
rule group_set grpset1 (time_length_factor 0.8)

define (group g5 (fromto U1-13 U3-10) (fromto U3-15 U4-7))
define (group g6 (fromto U1-9 U3-16) (fromto U3-14 U4-6))
define (group_set grpset2 g5 g6)
circuit group_set grpset2 (match_group_delay on (tolerance 400))
rule group_set grpset2 (time_length_factor .45)


set crosstalk_model cap_ratio
circuit class class1 (switch_window 10 35)
circuit class class2 (sample_window 1 25)
```

Circuit rules overview

Use the **circuit** command descriptors to

For length

- control maximum and minimum routed length of a net
- control maximum and minimum total routed length of all nets within a group
- match the routed length of each fromto to the longest fromto in a net or group
- match the routed lengths of all nets in a class
- match the routed length of each group in a group set

For delay

- control maximum and minimum delay for a net
- control the total delay for all nets within a group
- match the delays of each fromto in a net or group
- match the delays of all nets within a class
- match the total delay of each group in a group set

For noise and crosstalk

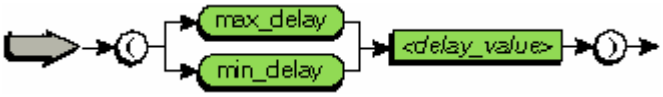
- control automatic shielding of wires during autorouting for a net, class, group, or fromto
- set the switch window and sample window for nets or classes of nets to define their net coupling relationship

For general autorouting

- Control routing priority
- Limit routing length on exposed layers
- Limit routing of specified nets and fromtos to certain layers
- Control which vias are used with certain nets, classes, or groups

<delay_descriptor>

The *<delay_descriptor>* sets a circuit rule that controls the maximum or minimum delay for a net.



max_delay

The maximum delay allowed. The routed length must be equal to or less than this value. If you enter a max_delay value that is less than the min_delay value, the max_delay value is ignored.

min_delay

The minimum delay allowed. The routed length must be equal to or greater than this value.

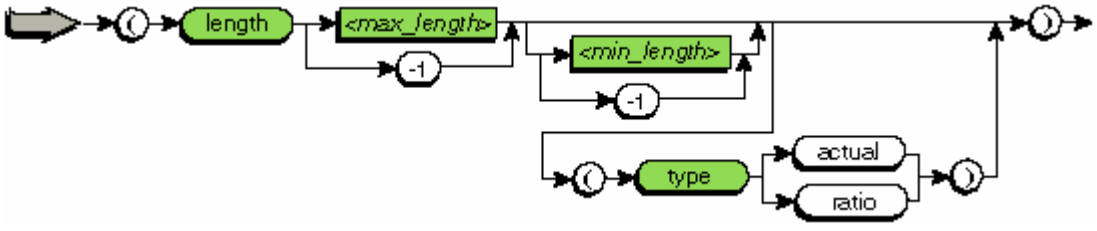
<delay_value>

The *<delay_value>* is a real number with up to three decimal places. For example, if you enter a value of 130.333333, the value is rounded off to 130.333.

If you specify a delay rule, you must also define how much delay each unit of routed wire length produces by setting a time_length_factor with the rule command. A delay rule is not applied unless you first set a time_length_factor.

<length_descriptor>

The *<length_descriptor>* sets a circuit rule that controls maximum and minimum routed wire lengths.



length

Controls maximum and minimum routed lengths.

<max_length>

The <*max_length*> value must be specified first, followed by the <*min_length*> value. Use a value of -1 to ignore a previously set maximum length value. If you enter a <*max_length*> that is less than the <*min_length*> value, <*max_length*> is ignored.

<min_length>

The <*min_length*> value is optional. If you don't want to control minimum length, omit the minimum length value. Use a value of -1 to ignore a previously set minimum length value. If you specify <*min_length*>, it must be less than the <*max_length*> value. If it is greater, <*max_length*> is ignored.

type

Controls whether the <*max_length*> and <*min_length*> values represent **actual** length values or a **ratio** of actual length to Manhattan length. If you don't specify **type**, the default is **actual**.

The <*max_length*> and <*min_length*> values can represent actual routed wire lengths or ratios of actual length to Manhattan length.

For example,

```
circuit net RX (length 1.25 1.1 (type ratio))
```

specifies a maximum routed wire length for net RX no greater than 125% and no less than 110% of its Manhattan length.

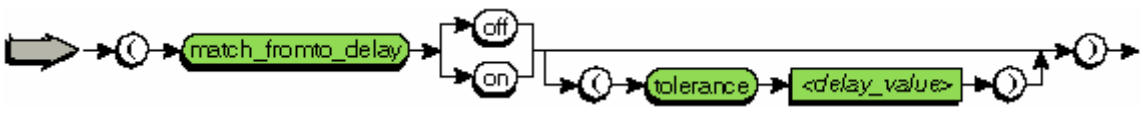
When **type ratio** is used, the <*max_length*> and <*min_length*> values use only two decimal places of precision. If you use more than two decimal places, the value is truncated. For example, the value 1.255 truncates to 1.25. The largest Manhattan length in a class is multiplied by the <*max_length*> and <*min_length*> factors to calculate the minimum and maximum length rules for all nets in the class.

Note

Device-level detailed placement follows length rules set with **type actual**, not **type ratio**.

<match_fromto_delay_descriptor>

The <*match_fromto_delay_descriptor*> sets a circuit rule that matches the delay of each fromto in a net or group.



match_fromto_delay

Matches the delay of each fromto in a net or group.

tolerance

The delays are matched within the tolerance specified with the command. The default delay tolerance is equal to a one inch actual dimension.

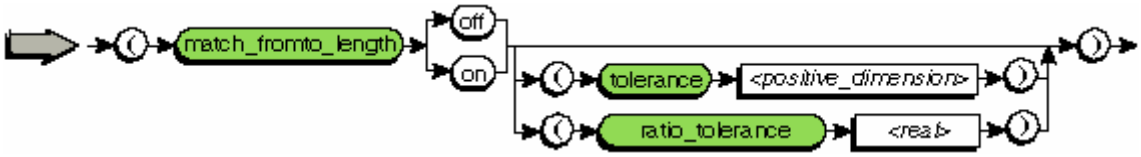
<delay_value>

The <*delay_value*> is a real number with up to three decimal places. For example, if you enter a value of 130.333333, the value is rounded off to 130.333.

If you specify a delay rule, you must define how much delay each unit of length produces by setting a time_length_factor with the rule command. A delay rule is not applied unless you first set a time_length_factor.

<match_fromto_length_descriptor>

The <*match_fromto_length_descriptor*> sets a circuit rule that matches the routed length of each fromto to the longest fromto in a net, group, or group set.



match_fromto_length

Matches the routed length of each fromto to the longest fromto in a net or group.

tolerance

The routed lengths are matched within the tolerance specified with the command, or within the default tolerance of one inch.

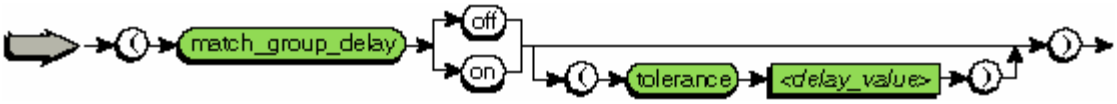
ratio_tolerance

The routed lengths are matched within the tolerance set as a percentage of the longest Manhattan length.

By default, the tolerance is set to one inch when match_fromto_length is on and no tolerance or ratio_tolerance is specified. When ratio_tolerance is specified, it must be a real number which is a percentage value with up to two decimal places. For example, if the ratio tolerance value is .20, and the longest fromto Manhattan length is 1.5 units, the tolerance is .3 units.

<match_group_delay_descriptor>

The <*match_group_delay_descriptor*> sets a circuit rule that matches the total delay of each group in a group_set. It applies only to defined group_sets.



match_group_delay

Applies only to a set of groups. Matches the total delay of each group in the set.

tolerance

The delays are matched within the tolerance specified with the command. The default delay tolerance is equal to a one inch actual dimension.

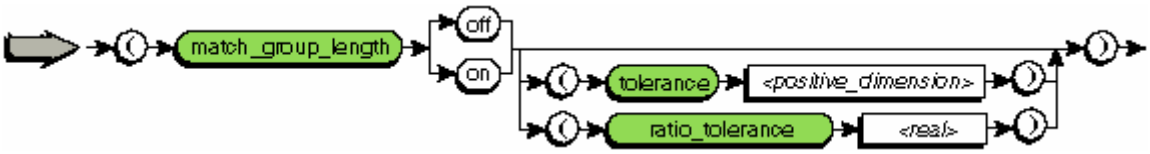
<delay_value>

The <*delay_value*> is a real number with up to three decimal places. For example, if you enter a value of 130.333333, the value is rounded off to 130.333.

If you specify a delay rule, you must define how much delay each unit of length produces by setting a time_length_factor with the rule command. A delay rule is not applied unless you first set a time_length_factor.

<match_group_length_descriptor>

The <*match_group_length_descriptor*> sets a circuit rule that matches the total routed length of each group in a group_set. It applies only to a group_set.



match_group_length

Applies only to a set of groups. Matches the total routed length of each group in the set.

tolerance

The routed lengths are matched within the tolerance specified with the command, or within the default tolerance of one inch.

ratio_tolerance

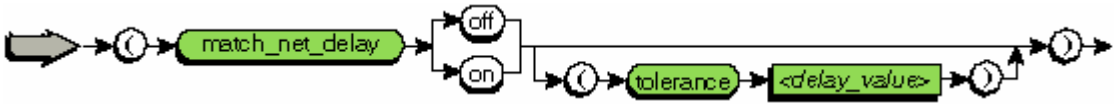
The routed lengths are matched within the tolerance set as a percentage of the longest Manhattan length.

By default, the tolerance is set to one inch when match_group_length is on and no tolerance or ratio_tolerance is specified. When ratio_tolerance is specified, it must be

a real number which is a percentage value with up to two decimal places. For example, if the ratio tolerance value is .20, and the longest total Manhattan length in the group_set is 1.5 units, the tolerance is .3 units.

<match_net_delay_descriptor>

The *<match_net_delay_descriptor>* sets a circuit rule that matches the delays of all nets in a class. It applies only to a class of nets.



match_net_delay

Applies only to a class of nets. Matches the delays of all nets in a class.

tolerance

The delays are matched within the tolerance specified with the command. The default delay tolerance is equal to a one inch actual dimension.

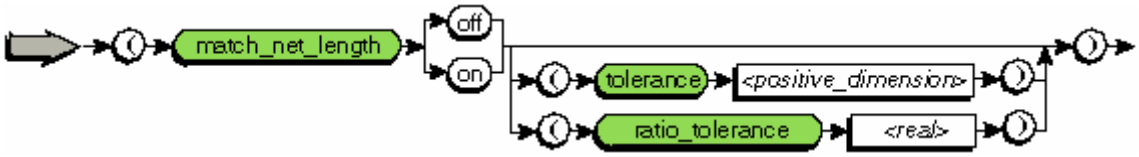
<delay_value>

The *<delay_value>* is a real number with up to three decimal places. For example, if you enter a value of 130.333333, the value is rounded off to 130.333.

If you specify a delay rule, you must define how much delay each unit of length produces by setting a time_length_factor with the rule command. A delay rule is not applied unless you first set a time_length_factor.

<match_net_length_descriptor>

The *<match_net_length_descriptor>* sets a circuit rule that matches the routed lengths of all nets in a class. It Applies only to a class of nets.



match_net_length

Applies only to a class of nets. Matches the routed lengths of all nets in a class.

tolerance

The routed lengths are matched within the tolerance specified with the command, or within the default tolerance of one inch.

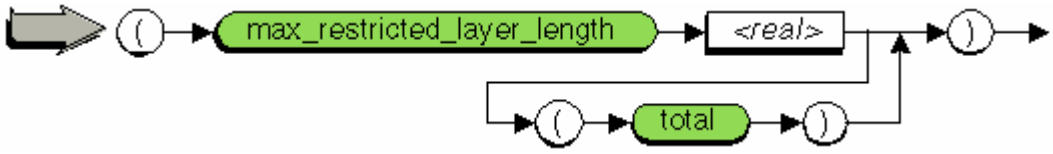
ratio_tolerance

The routed lengths are matched within the tolerance set as a percentage of the longest Manhattan length.

By default, the tolerance is set to one inch when match_net_length is on and no tolerance or ratio_tolerance is specified. When ratio_tolerance is specified, it must be a real number which is a percentage value with up to two decimal places. For example, if the ratio tolerance value is .20, and the longest total Manhattan length in the group_set is 1.5 units, the tolerance is .3 units.

<max_restricted_layer_length_descriptor>

The <*max_restricted_layer_length_descriptor*> sets a circuit rule that limits routed length on restricted layers. This circuit rule applies to nets, classes of nets, fromtos, groups, and group sets.



max_restricted_layer_length

Sets a maximum routed length (<>) for individual nets and fromtos on restricted layers.

total

Applies to groups only. This option limits the total routed length of fromtos in a group on restricted layers.

This rule is provided to limit routing on exposed layers. It works in conjunction with the *<restricted_layer_length_factor_descriptor>* which marks a layer as restricted.

For example,

rule layer sig1 sig4 (restricted_layer_length_factor 1)

marks layers sig1 and sig4 as restricted, and then

circuit class all_nets (max_restricted_layer_length 50)

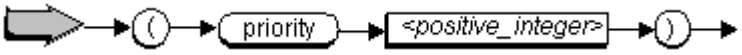
limits each net in the class all_nets to a maximum of 50 mils on layers sig1 and sig4.

Note

At the class and group set levels this rule applies to individual nets and groups, respectively.

<priority_descriptor>

The <*priority_descriptor*> affects when a net, class, or fromto is scheduled for routing.



The value of <*positive_integer*> can be any integer value in the range of 1 (lowest priority) to 255 (highest priority). When **priority** is not specified, nets have the default priority value of 10.

<sample_window_descriptor>

The <*sample_window_descriptor*> sets a circuit rule that defines one or more portions of a clock cycle during which sampling of the specified signals can occur.

A sample window is defined by a pair of integers that indicate the beginning and ending points of the window, with respect to the full master clock cycle. The sample window specifies the portion of the master clock cycle during which a net is susceptible to switching noise from an adjacent net.

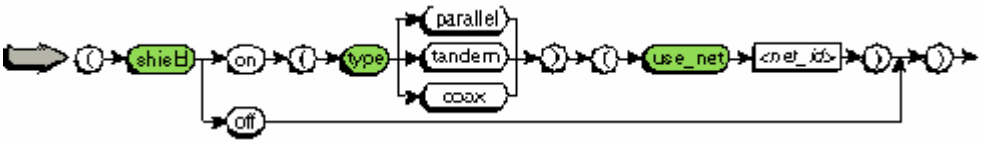
A value of -1 entered immediately after the **sample_window** keyword removes the entire sample window definition and leaves it unspecified.

Switch/Sample Window overlap

Once switch and sample windows are defined, noise transmission and reception for nets are determined based on whether their defined switch and sample window intervals overlap. For example, if a switch window for net A overlaps with the sample window of net B, then net B may receive switching noise transmitted from net A unless the nets are routed in compliance with other noise and crosstalk rules. In other words, nets will be routed according to noise and crosstalk rules where switch/sample window overlaps indicate a noise transmission/reception may occur.

<shield_descriptor>

The <*shield_descriptor*> sets a circuit rule that controls whether shielding is applied to wires.



shield

Controls whether a connection is automatically shielded during autorouting.

type

Specifies one of three shield types:

parallel, which allows parallel shield wires on the same layer for nets with this rule.
tandem, which allows parallel shield wires on layers above and below nets with this rule.
coax, which combines **parallel** and **tandem** to allow shield wires on the same and adjacent layers for nets with this rule.

The default is **parallel**.

use_net

The **use_net** <*net_id*> syntax specifies the shield net, which must be a net assigned as a power layer in the design.

Use this descriptor only to control automatic shield creation for a net, class, group, or fromto. By default, this descriptor allows parallel shields. The **tandem** keyword allows shields on layers above and below the shielded wire. The **coax** keyword allows both tandem and parallel shields. Shields are routed during automatic and interactive routing of nets that have this shield rule.

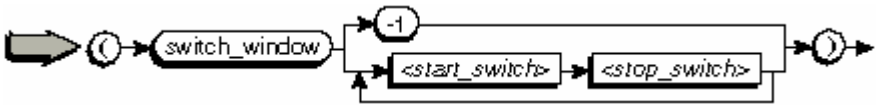
See also

Related tandem shield rules:

<tandem_shield_overhang_descriptor>

<switch_window_descriptor>

The <*switch_window_descriptor*> sets a circuit rule that defines one or more portions of a clock cycle during which switching of the specified signals occurs.



A switch window is defined by a pair of integers that indicate the beginning and ending points of the window, with respect to the full master clock cycle. The switch window specifies the portion of the master clock cycle during which a net may transmit switching noise to an adjacent net.

A value of -1 entered immediately after the **switch_window** keyword removes the entire switch window definition and leaves it unspecified.

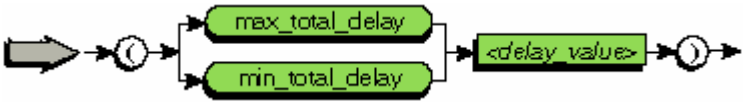
Switch/Sample Window overlap

Once switch and sample windows are defined, noise transmission and reception for nets are determined based on whether their defined switch and sample window intervals overlap. For example, if a switch window for net A overlaps with the sample window of net B, then net B may receive switching noise transmitted from net A unless the nets are routed in compliance with other noise and crosstalk rules. In other words, nets will be routed according to noise and crosstalk rules where switch/sample

window overlaps indicate a noise transmission/reception may occur.

<total_delay_descriptor>

The <*total_delay_descriptor*> sets a circuit rule that controls the range for the total delay of a group. The rule applies only to groups.



max_total_delay

Applies only to groups. The **max_total_delay** rule sets the range for the total delay of a group.

min_total_delay

Applies only to groups. The **min_total_delay** rule sets the range for the total delay of a group.

<delay_value>

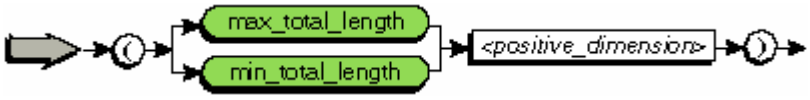
The <*delay_value*> is a real number with up to three decimal places. For example, if you enter a value of 130.333333, the value is rounded off to 130.333.

The sum of the delays of the routed fromtos in a group must be equal to or less than the max_total delay and equal to or greater than the min_total_delay.

If you specify a delay rule, you must define how much delay each unit of length produces by setting a time_length_factor with the rule command. A delay rule is not applied unless you first set a time_length_factor.

<total_length_descriptor>

The <*total_length_descriptor*> sets a circuit rule that controls the maximum and minimum limits for the total routed length of fromtos in a group. The rule applies only to groups.



max_total_length

Applies only to groups. The **max_total_length** rule sets the maximum total length of a group.

min_total_length

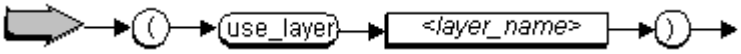
Applies only to groups. The **min_total_length** rule sets the minimum total length of a

group.

The sum of the routed lengths of the fromtos in the group must be equal to or less than the **max_total_length** and equal to or greater than the **min_total_length**.

<use_layer_descriptor>

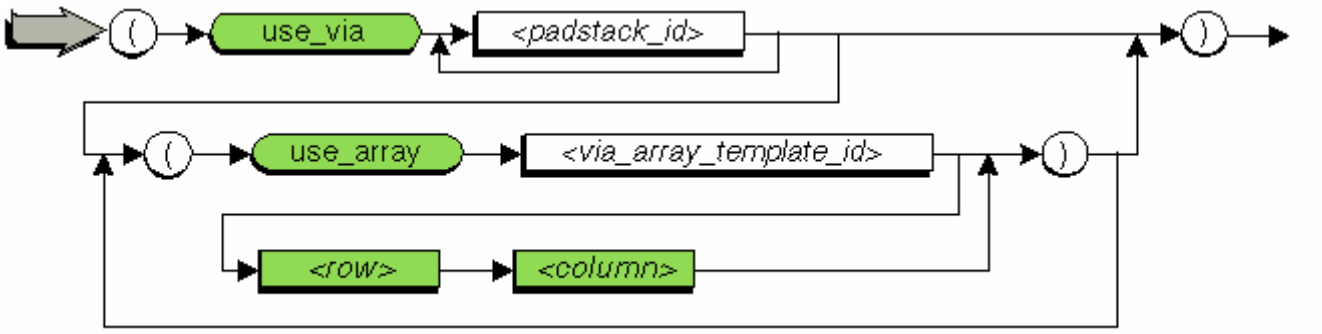
The <*use_layer_descriptor*> assigns one or more routing layers to a net, class, group, group set, or fromto.



Each <*layer_name*> is the name of a layer on which the net(s) or fromto(s) can be routed. The **use_layer** rule overrides an unselected layer.

<use_via_descriptor>

The <*use_via_descriptor*> sets a circuit rule that is used during autorouting. The use_via or use_array rule can apply to named nets, classes, groups, or to selected nets.



use_via

The **use_via** rule assigns one or more vias (<*padstack_id*>) or a via_array (<*via_array_template_id*>) to a net, class, group, group set, fromto, or to selected nets. When you assign more than one via, the autorouter chooses the padstack with the smallest shape that satisfies the layer requirements.

You can substitute a <*via_array_template_id*> for a <*padstack_id*>, if you want to use a via array but do not want to specify the number of rows and columns of the array.

use_array

The **use_array** option generates a via array (<*via_array_template_id*>) from information in the via array template. You must specify the via array size with the **row** and **column** parameters. The **use_array** option overrides the default via even when the wire intersection area is only large enough for a single via.

For more information about via arrays, see the <*via_array_template_descriptor*> in the *Design Language Reference*.

Note

Via array features are only available with the MicroVia option, which requires the RouteMVIA license.

<row>

Number of rows in the via array.

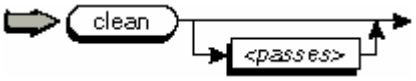
<column>

Number of columns in the via array.

The use_array rule identifies the template you want to use for via arrays. The minimum number of vias used to interconnect wires when you specify use_array is the number defined in the template. The maximum number of vias applied when you specify use_array is based on the number that fit in the area where interconnecting wires overlap. Wider wires can use more vias to improve connectivity

clean

The **clean** command initiates rip-up and reroute passes that improve manufacturability by removing vias and bend points and by changing SMD entries and exits.



Clean passes improve PCB manufacturability. The **clean** command rips-up and reroutes all connections with higher costs for parameters that include via use, off-center SMD pad entry, and SMD pad side-exit. The use of **clean** results in better quality routes. Four clean passes are suggested after completing all routing passes. If you use the command without a pass value, the autorouter performs one clean pass.

The routing progress indicator monitors and displays the progress of the **clean** command using a traffic light icon. You can click on the icon to display detailed information in a dialog box.

Note

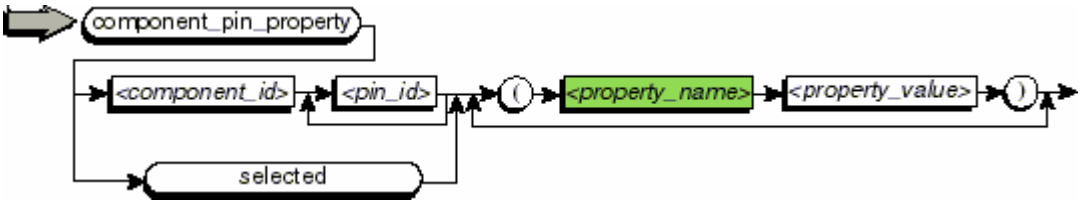
The **clean** command should not be executed after a **miter** command.

Command examples

clean
clean 4

component_pin_property

The **component_pin_property** command assigns properties to component pins.



<property_name>

A keyword that identifies a standard property or a user property. Each property you assign must consist of a keyword (<*property_name*>) and a value (<*property_value*>). The value might be another keyword, a number, or a character string depending on what the property requires.

This command lets you assign both standard properties and user properties to one or more pins on a component. You can specify the component name (<*component_id*>) and each pin name (<*pin_id*>), or you can use the **selected** option to apply the property to all selected pins.

A property consists of the keyword (<*property_name*>) that identifies the property, and a value (<*property_value*>). Property values can be numbers, keywords, or character strings depending on the property.

The standard properties for component pins include

force_to_terminal_point <*property_value*>
exit_direction <*property_value*>

Properties can be assigned in SPECCTRA or in the design file, but a property assigned to a pin in the design file cannot be changed or removed in SPECCTRA. Component pin properties apply only to individual pins on specific component instances of an image. A property assigned to a component pin takes precedence over a property assigned to the image pin.

You can use the report command to generate a property report that contains the current values of properties assigned to all component pins in the design.

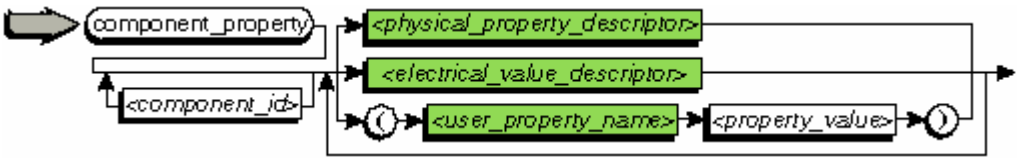
Command examples

component_pin_property C81 2 (uprop_1 0.02)

component_pin_property I6301 3 5 9 (uprop_2 xyz)

component_property

The **component_property** command assigns physical, electrical, and user properties to components.



This command lets you assign both standard properties and user properties to one or more components. The standard component properties consist of several physical properties and an electrical value. Physical properties consist of type, height, and power dissipation.

In general, a property consists of the keyword (<*property_name*>) that identifies the property, and a value (<*property_value*>). Property values can be numbers, keywords, or character strings depending on the property. See component properties for a list of properties you can assign to components.

You can either select the components before using this command or specify the reference designator (<*component_id*>) for each component. If you do not specify component reference designators, SPECCTRA assigns the properties to all selected components.

Properties can be assigned in SPECCTRA or in the design file. Component properties apply only to specific component instances of an image. A property assigned to a component takes precedence over a property assigned to the component's image. Use the image_property command to assign properties to images.

The standard component properties consist of physical and electrical properties.

- The physical properties let you control a component's type, maximum height, and maximum power dissipation.
- The electrical property is a label you can assign that identifies some electrical characteristic of a component.

You can use the report command to generate a property report that contains the current values of properties assigned to all components in the design. You can also generate a total power dissipation report for the PCB.

Note

If you assign or remove physical or electrical properties on components, SPECCTRA records these changes when you use the write command to save a placement file or a session file. User properties assigned to or removed from components, and physical, family, and user properties assigned to images (using **image_property**) or removed from images, are not recorded in these files.

See also

autodiscrete
autorotate
define room
initplace
interchange
place_rule

room_rule
select component
unplace

Command examples

The following examples assign properties to the named components.

component_property C81 (type capacitor) (height 0.0280)

component_property U28 U40 (height 0.1800)

component_property R1 R2 R5 (power_dissipation 500)

component_property U1 U2 (value 10k)

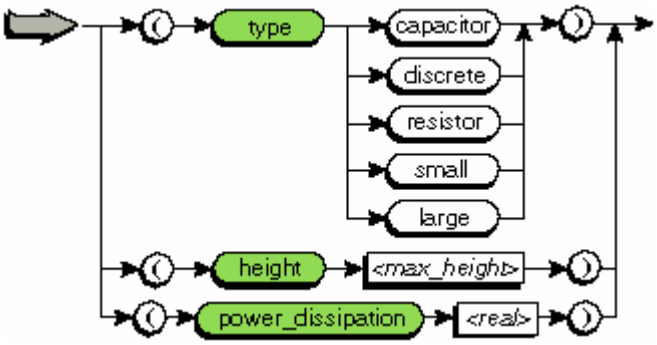
The following examples assign properties to all selected components.

component_property (type capacitor) (value 0.5pf) (power_dissipation 0.5)

component_property (height 0.05)

<physical_property_descriptor>

Use <*physical_property_descriptor*> to assign type, height, and power dissipation properties to components or images.



type

Controls which small components are included for processing in the current automatic placement operation. A small component is a component with three pins or less that has not been assigned the large type property. The choices are

capacitor, which includes only small capacitors (small components assigned the capacitor type property, and small components with all pins connected to power nets and not assigned the resistor or discrete type property).

discrete, which includes only small discretes (small components assigned the discrete type property).

resistor, which includes only small resistors (small components assigned the resistor type property).

small, which includes all small components.

The default is **small**.

height

Assigns maximum and minimum component height constraints for a room. A value of -1 for <*max_height*> or <*min_height*> means that height constraint is undefined. The defaults are both -1.

power_dissipation

Assigns a maximum power dissipation value for total dissipation of all components in the room. A value of -1 means the power dissipation constraint is undefined. The default is -1.

The physical properties you can assign to a component or image consist of one or two types (**type**), maximum height (**height**), and maximum power dissipation (**power_dissipation**). You can assign or change any or all of these properties in the same command.

- Use **type** when you want to classify components for placement rules or for exclusive processing in automatic placement operations.
- Use **height** when you plan to constrain the minimum or maximum height of components permitted in a room.
- Use **power_dissipation** when you plan to constrain the maximum total power dissipation permitted in a room.

SPECCTRA recognizes the following component and image types:

- Large
- Small
- Capacitor
- Resistor
- Discrete

By default, a large component or image has more than three pins, and a small component has three pins or less. The large and small types are mutually exclusive. Assigning one of them removes the other. You can assign the large type to a component or image with three pins or less, but you cannot assign the small type to a component or image with more than three pins.

You can assign the capacitor, resistor, or discrete type to any small or large component or image. These types are mutually exclusive. Assigning one of them to a component or image removes either of the others.

A capacitor in SPECCTRA is defined as a decoupling (bypass) capacitor. If a component with three or fewer pins, all connected to power nets, has not been assigned the large, resistor, or discrete type, SPECCTRA automatically treats the component as a capacitor.

SPECCTRA distinguishes between large and small components for processing in automatic placement operations. You can also specify small capacitors, resistors, or discretes for exclusive processing. Large capacitors, resistors, or discretes must be processed with other large components.

You can assign separate image set placement rules for each type on the PCB or within a room. Capacitor, resistor, or discrete type rules take precedence over large or small type rules. See place_rule for details.

See also general information about component and image types.

Note

See the define room and room_rule commands for details about setting placement constraints for rooms.

If you assigned jumper heights to jumpers in the design file and you want to route jumpers beneath components, you must assign to each component (or its image) a **height** property with a value that is greater than any jumper height assigned to jumpers in the design file.

<electrical_value_descriptor>

Use <*electrical_value_descriptor*> to assign an electrical value to individual components.

The electrical value property is a label that identifies an electrical part or characteristic but has no functional significance in SPECCTRA.

<user_property_name>

A keyword that identifies a user property. Each property you assign must consist of a keyword (<*property_name*>) and a value (<*property_value*>). The value might be another keyword, a number, or a character string depending on what the property requires.

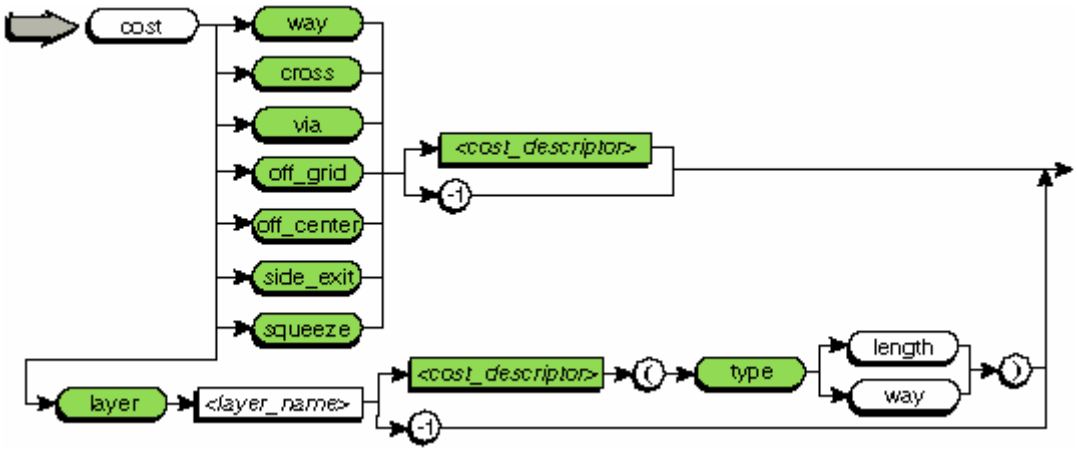
A user property is treated as a label in SPECCTRA, but can have functional meaning to the host layout system or a third party tool.

cost

The **cost** command sets routing costs and overrides the autorouter internal cost table.

Note

The **tax** command is preferred over **cost** if you need to apply routing costs.



way

The cost to route in the wrong direction. For example, the cost of horizontal wire segments routed on a vertical layer.

cross

The cost of a crossing conflict.

via

The cost to use a via.

off_grid

The cost to route off grid. The autorouter routes off grid unless you use the command **cost off_grid forbidden**. If you use gridless routing, this cost does not apply.

off_center

The cost to enter or exit a pin off center.

side_exit

The cost to exit pins on the long side.

squeeze

The cost to create a wire-to-via clearance violation.

<cost_descriptor>

You can set values for cost options with the <*cost_descriptor*>. The <*cost_descriptor*> can be a keyword or a numeric value. The cost descriptors and their corresponding numeric values are listed in the following table.

Cost Description	Numeric Value
forbidden	100

high	50
medium	25
low	8
free	0

layer

The cost to use a named layer (<*layer_name*>) for routing, controlled by **type**, which is either **length** or **way**.

type

Controls how the cost applies on the named layer

length - the cost of any routing on the layer

way - the cost of wrong-way routing on the layer

You can override internally defined costs and set them to fixed values with this command, although this is not generally recommended. If you don’t use the **cost** command, the autorouter automatically adjusts costs throughout the autorouting session, using default costs.

When you execute a **cost** command, the cost value you specify remains constant until you change it or pass control back to the autorouter by resetting the value to -1. If you want to return a cost parameter to its default (system-assigned) value, execute the **cost** command with a value of -1. For example:

```
cost way -1
```

You can set values for cost options with the <*cost_descriptor*>. The <*cost_descriptor*> can be a keyword or a numeric value. The cost descriptors and their corresponding numeric values are listed in the following table.

Cost Description	Numeric Value
forbidden	100
high	50
medium	25
low	8
free	0

When you set a cost to **forbidden**, the autorouter is not prohibited from overriding that cost except for vias. If you execute **cost via forbidden**, the autorouter is prohibited from using vias. A more efficient way of prohibiting vias is to **unselect all vias**.

The **cost off_grid forbidden** command is ignored when the center of an SMD pad is off-grid. The autorouter must go off-grid to route an off-grid pad.

Command examples

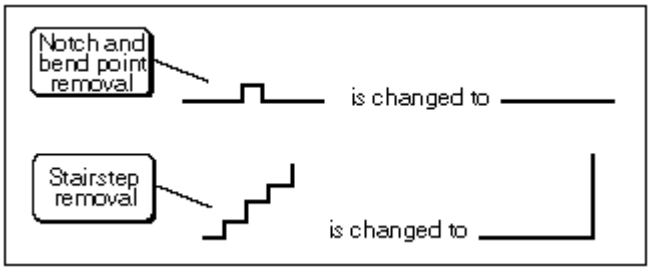
cost way forbidden
cost via low
cost layer L1 forbidden
cost layer L2 high (type way)

critic

The **critic** command helps improve manufacturability without performing a rip-up and reroute operation.

The **critic** command inspects the routing to eliminate notches and removes extra bends. The **critic** command is similar to the **clean** command but different in one important respect. Where **clean** completely reroutes each wire and can drastically change a connection's wiring, the **critic** command attempts to make local adjustments to the existing wires without rip-up and rerouting. The **critic** operation executes much faster than **clean**.

The following figure shows notch and bend point removal and stairstep removal.



See also

clean

Command example

critic

Define Commands

Define commands create classes, groups, group sets, differential pairs, bundles, and regions. These commands can also be used to define net ordering, class-to-class relationships, and layer noise weight factors.

Define commands also allow you to use rule and circuit descriptors to assign rules to nets, classes, groups, group sets and fromtos., and differential net pairs. You can create a table of layer noise weight factors to represent your design's layer-to-layer

noise coupling characteristics. Refer to the *Design Language Reference* manual for additional information on layer_noise_weight.

You can use the fromto descriptor to define net ordering, assign layer rules, and apply rule and circuit commands. See the *<fromto_descriptor>* for more detailed information.

The individual **define** commands are explained in the following topics. Syntax diagrams and command examples are included.

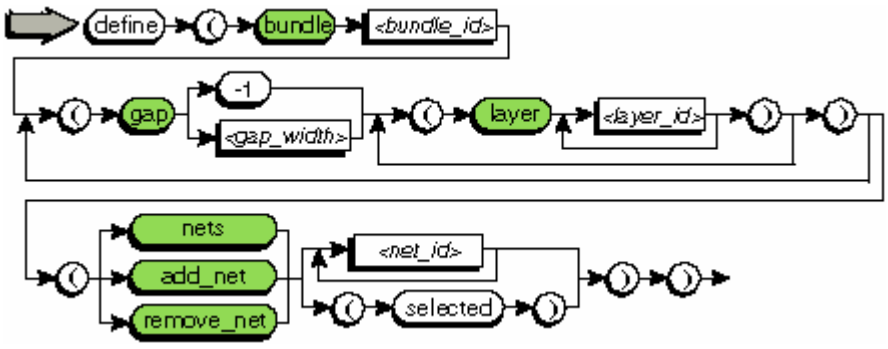
define bundle
define class
define class_class
define group
define group_set
define layer_noise_weight
define net
define padstack
define pair
define region

Note

For information about defining keepout areas, see the define keepout command.

define bundle

The **define bundle** command assigns nets to named bundles for later routing with similar path topologies.



bundle

Creates or edits a net bundle that includes information about intended spacing and layers for later routing with the same path topology.

gap

Specifies the intended spacing between wires routed as a bundle. The **gap** can apply to one or more layers, and multiple gaps can be specified.

layer

Identifies one or more layers on which the specified **gap** applies.

nets

Identifies nets by <*net_id*> or uses **selected** nets.

add_net

Adds one or more nets to an existing net bundle. You can use this option without the **gap** option.

remove_net

Removes one or more nets from an existing net bundle. You can use this option without the **gap** option.

Note: It is possible to remove all nets from a net bundle, leaving an empty net bundle to which nets may be added using the **add_net** option.

Use this command to prepare two or more nets for routing with the same path topology. A net may only belong to one bundle at a time. After a bundle has been defined, nets can be removed and added to the bundle using the **remove_net** and **add_net** options.

The order of the <*net_id*>s or selected nets in this command has no effect on the routing order of the bundled nets. Routing order depends on the physical layout of the pins of the bundled nets.

Command examples

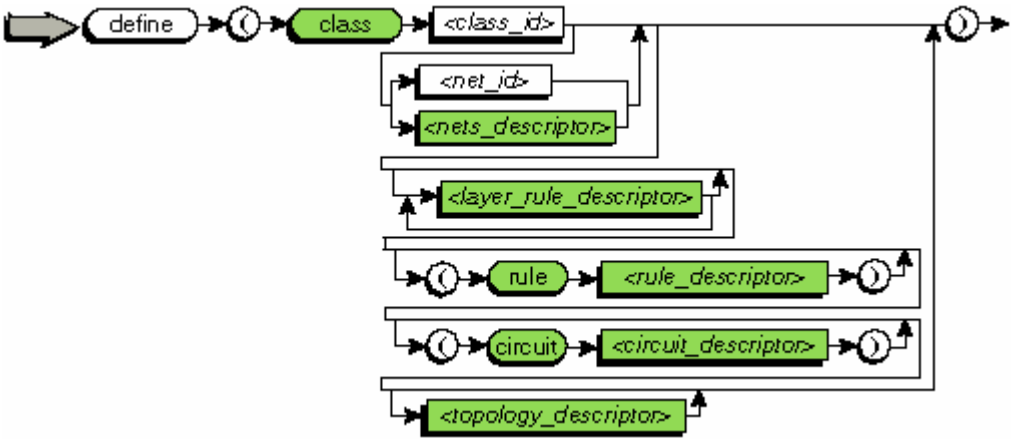
define (bundle addr_bundle (gap 10 (layer m1 m2)) (nets addr1 addr2 addr3))

define (bundle addr_bundle (add_net addr4))

define (bundle addr_bundle (remove_net addr1))

define class

The **define class** command assigns a name to a group of nets. Optionally, it also can assign rules to the class it defines.



class

A group of nets that are referenced by a single name.

rule

Assigns one or more rules in the current command. Click *<rule_descriptor>* to see which rules apply for this command.

circuit

Assigns one or more circuit rules in the current command. Click *<circuit_descriptor>* to see which rules apply for this command.

You can use the <*rule_descriptor*> and <*circuit_descriptor*> to apply clearance, wiring, timing, crosstalk, and noise rules to classes.

When you use the **define class** command, consider the following guidelines and restrictions:

- Class names must be unique.
- You can assign a net to more than one class, but if the classes have conflicting rules, the rule of the last defined class is used.

Class definition

A defined class is available for assignment of a variety of rules that will apply to all the nets in the class, according to the rules hierarchy. You assign rules to an existing class with the **circuit** and **rule** commands. To save a step, you can assign rules when you define the class, using the rule descriptor and circuit descriptor within the define class command.

Adding nets to a class

The add_net and add_selected_nets options add nets to an existing class. Nets already in the class remain and existing rules apply to the added nets.

Removing nets from a class

The remove_net and remove_selected_nets options remove one or more nets from

an existing class without disbanding the class. Nets not specified when using this option remain in the class, and all rules currently assigned to the class remain in effect.

Note

You can redefine the rules of an existing class by omitting the net name list, and specifying the new rules for the class.

See also

circuit and rule commands for complete <*circuit_descriptor*> and <*rule_descriptor*> diagrams and descriptions.

forget command to disband a class

Command examples

The following example creates a class named “c2” consisting of three nets.

```
define (class c2 sig2 sig3 sig4)
```

The next example creates a class named “c3” and assigns a circuit rule to it.

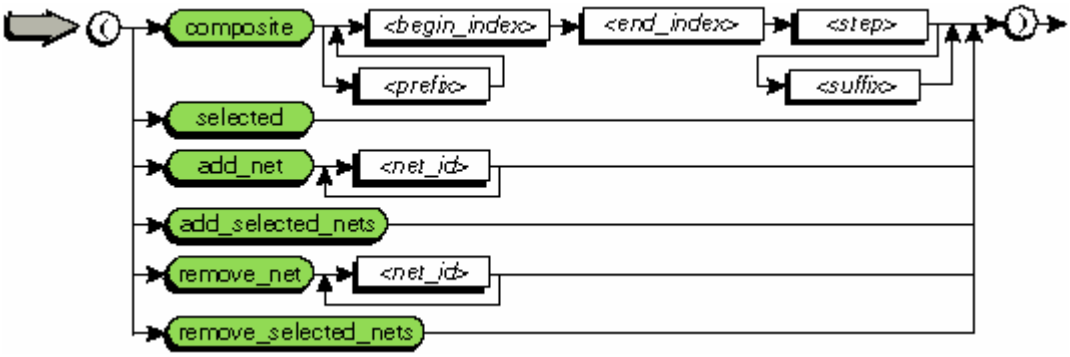
```
define (class c3 sig5 sig6 (circuit (use_via v25)))
```

This example creates a class named “c4” and assigns two rules to it: a width rule and a clearance rule.

```
define (class c4 sig7 sig8 (rule (width 0.010) (clearance 0.008)))
```

<*nets_descriptor*>

The <*nets_descriptor*> names the nets in the class.



composite

Identifies a list of net names that match the <*begin_index*>, <*end_index*>, <*step*>, and optional <*prefix*> and <*suffix*> parameters.

<*prefix*>	One or more non-numeric characters that match the initial characters of one or more nets. The <*prefix*> parameter cannot include wildcard characters.

<*begin_index*>	A positive integer that matches the integer portion of a net name. The <*begin_index*> parameter determines the initial integer to match in a range.
<*end_index*>	A positive integer that matches the integer portion of a net name. The <*end_index*> parameter determines the last integer to match in a range.
<*step*>	A positive integer that determines which integer values to match in net names between the <*begin_index*> and <*end_index*>parameters.
<*suffix*>	One or more non-numeric characters that match the ending characters of one or more net names. The <*suffix*> parameter cannot include wildcard characters.

selected

Includes selected nets in the specified class.

add_net

Adds one or more specified nets to the named class.

add_selected_nets

Adds currently selected nets to the named class.

remove_net

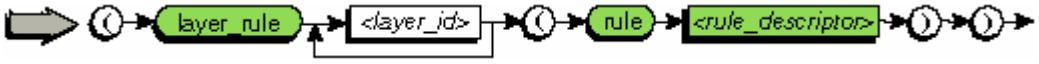
Deletes one or more specified nets from the named class without disbanding the class.

remove_selected_nets

Deletes currently selected nets from the named class without disbanding the class.

<layer_rule_descriptor>

The <*layer_rule_descriptor*> assigns a layer rule to the defined class.



layer_rule

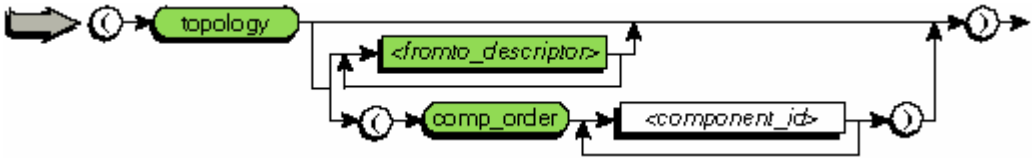
A routing rule that applies to all wires routed on the named layers, unless a higher-precedence rule overrides.

rule

Assigns one or more rules in the current command. Click *<rule_descriptor>* to see which rules apply for this command.

<topology_descriptor>

The *<topology_descriptor>* defines the preferred ordering of pin connections for each net in the class.



topology

Defines the preferred topology, which is the exact ordering of pin connections for each member net of the class.

<fromto_descriptor>

Defines each single pin-to-pin connection for member nets of a class. See the *<fromto_descriptor>* for a complete diagram and description.

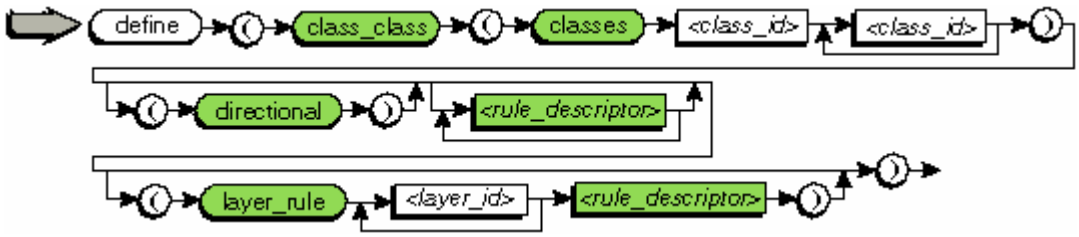
Fromtos cannot be identified with a *<pin_reference>* in a class.

comp_order

Orders the pin-to-pin connections of a net by using the component reference designator *<component_id>* only.

define class_class

The **define class_class** command assigns a name to a group of two or more classes for the purpose of assigning inter-class rules to the classes.



class_class

Defines a group of classes that can be referenced by a single name.

classes

The name of a class of nets that's defined in the design file or by using the define class command.

At least two class ids must be supplied. All classes are paired with each other when multiple classes are listed. To define rules between specific classes, you specify separate **class_class** commands, listing only the two classes to be paired. You can repeat a class id to apply rules between the wires of that class only.

directional

The **directional** keyword determines which class is noise transmitter or noise receiver. Direction is used only for *parallel noise descriptors* and *tandem noise descriptors*. The rule applies to the pair in the order the classes are specified. Do not use **directional** when applying crosstalk rules between the wires of a single class.

layer_rule

A routing rule that applies to all wires routed on the named layers, unless a higher-precedence rule overrides.

You can apply clearance, crosstalk, and noise rules to classes for class_class.

See also

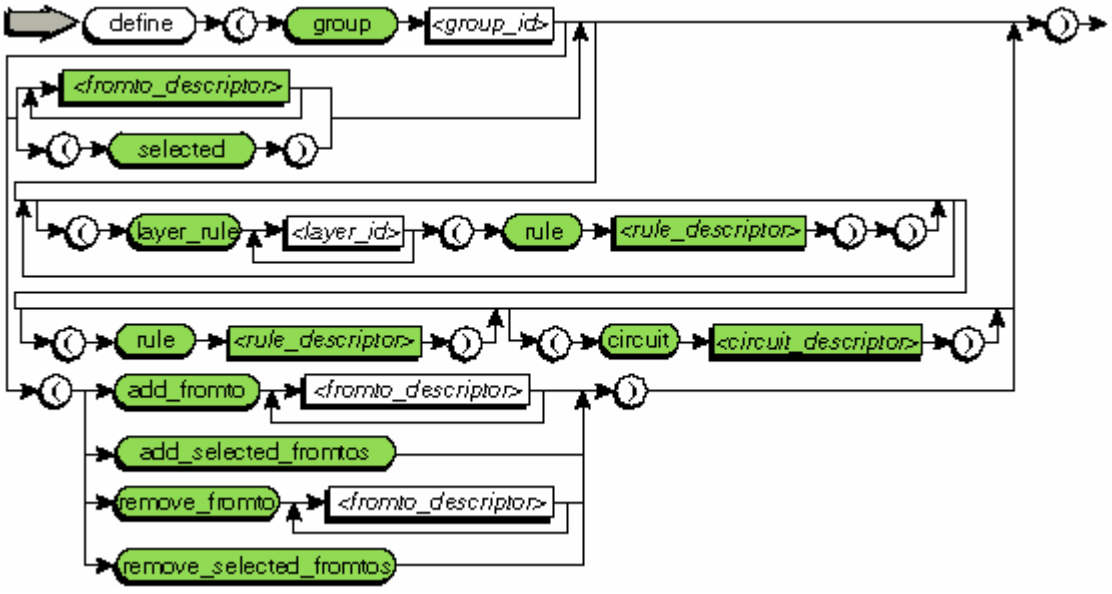
rule command for complete <*rule_descriptor*> diagrams and descriptions.

Command examples

define (class_class C2 C3 (rule (parallel_segment (gap 0.005) (limit 0.050))))

define group

The **define group** command assigns a name to a group of fromtos. Optionally, it can also assign one or more circuit and routing rules to the group it defines.



group

A group of fromtos that are referenced by a single name <*group_id*>.

<fromto_descriptor>

Defines one or more pin-to-pin connections as members of a group. See the *<fromto_descriptor>* for a complete diagram and description.

You can use the net name to differentiate virtual pins in groups, since virtual pin names are not unique.

selected

Includes currently selected fromtos in the group.

layer_rule

A routing rule that applies to all wires routed on the named layers, unless a higher-precedence rule overrides.

rule

Assigns one or more rules in the current command. Click *<rule_descriptor>* to see which rules apply for this command.

circuit

Assigns one or more circuit rules in the current command. Click *<circuit_descriptor>* to see which rules apply for this command.

add_fromto

Adds one or more fromtos, using the fromto descriptor, to the named group.

add_selected_fromtos

Adds currently selected fromtos to the named group.

remove_fromto

Deletes one or more specified fromtos, using the fromto descriptor, from the named group without disbanding the group.

remove_selected_fromtos

Deletes currently selected fromtos from the named group without disbanding the group.

You can apply clearance, wiring, timing, shielding, crosstalk, and noise rules to groups.

Group definition

A defined group is available for assignment of a variety of rules that will apply to all the fromtos in the group, according to the rules hierarchy. You assign rules to an existing group with the **circuit** and **rule** commands. To save a step, you can assign rules when you define the group, using the rule descriptor and circuit descriptor within the **define group** command.

Adding fromtos to a group

The add_fromto and add_selected_fromtos options add fromtos to an existing group. Fromtos already in the group remain and existing rules apply to the added fromtos.

Removing fromtos from a group

The remove_fromto and remove_selected_fromtos options remove one or more fromtos from an existing group without disbanding the group. Fromtos not specified when using this option remain in the group, and all rules currently assigned to the group remain in effect.

Note

You can assign a fromto to more than one group, but if the groups have conflicting rules, the rule of the last defined group is used.

You can redefine the rules of an existing group by omitting the fromto list, and specifying the new rules for the group.

See also

circuit and rule commands for complete <*circuit_descriptor*> and <*rule_descriptor*> diagrams and descriptions

forget command to disband groups

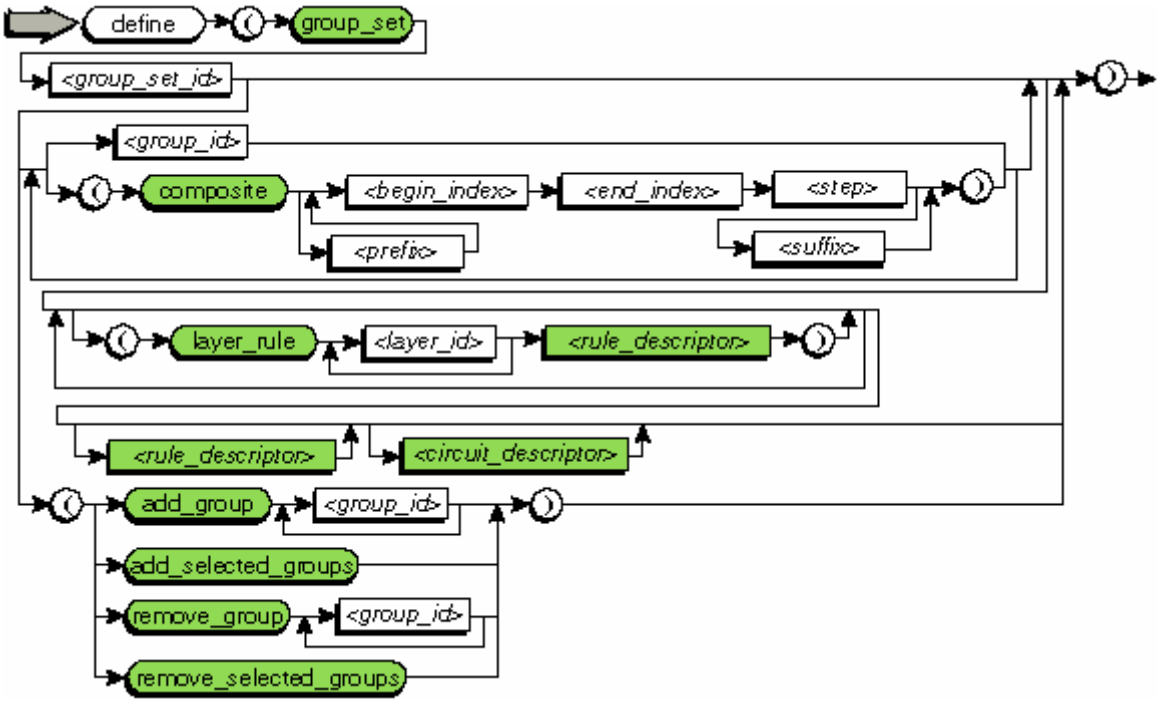
Command examples

```
define (group g1
   (fromto U1-20 U2-33)
   (fromto U3-12 U4-16)
   (fromto U5-1 U6-4)
   (circuit (use_via v12)))
```

```
define (group g1
   (add_fromto (fromto U1-21 U2-34))
```

define group_set

The **define group_set** command assigns a name to a number of specified groups. Optionally, it can also assign routing and circuit rules that apply to all fromtos in the groups that make up the group set.



group_set

A set of groups that are referenced by a single name <*group_set_id*>.

composite

Identifies a list of group names that match the <*begin_index*>, <*end_index*>, <*step*>, and optional <*prefix*> and <*suffix*> parameters.

<*prefix*>	One or more non-numeric characters that match the initial characters of one or more groups. The <*prefix*> parameter cannot include wildcard characters.
<*begin_index*>	A positive integer that matches the integer portion of a group name. The <*begin_index*> parameter determines the initial integer to match in a range.
<*end_index*>	A positive integer that matches the integer portion of a group name. The <*end_index*> parameter determines the last integer to match in a range.
<*step*>	A positive integer that determines which integer values to match in group names between the <*begin_index*> and <*end_index*>parameters.
<*suffix*>	One or more non-numeric characters that match the ending characters of one or more group names. The <*suffix*> parameter cannot include wildcard characters.

layer_rule

A routing rule that applies to all group_sets on the named layers, unless a higher-precedence rule overrides.

add_group

Adds one or more existing groups to the named group set.

add_selected_groups

Adds currently selected groups to the named group set.

remove_group

Deletes one or more specified groups from the named group set without disbanding the group set.

remove_selected_groups

Deletes currently selected groups from the named group set without disbanding the group set.

You can apply clearance, width, and timing rules to group sets.

Group set definition

A defined group set is available for assignment of a variety of rules that will apply to all the fromtos in all the groups of the set, according to the rules hierarchy. You assign rules to an existing group set with the **circuit** and **rule** commands. To save a step, you can assign rules when you define the group set, using the rule descriptor and circuit descriptor within the **define group_set** command.

Adding groups to a group set

The add_group and add_selected_groups options add already-defined groups to an existing group set. Groups already in the group set remain and existing rules extend to all the fromtos in the added groups.

Removing groups from a group set

The remove_group and remove_selected_groups options remove one or more groups from an existing group set without disbanding the group set. Groups not specified when using this option remain in the group set, and all rules currently assigned to the group set remain in effect.

Note

You can assign a group to more than one group set, but if the group sets have conflicting rules, the rule of the last defined group set is used.

See also

circuit and rule commands for complete <*circuit_descriptor*> and <*rule_descriptor*>

diagrams and descriptions.

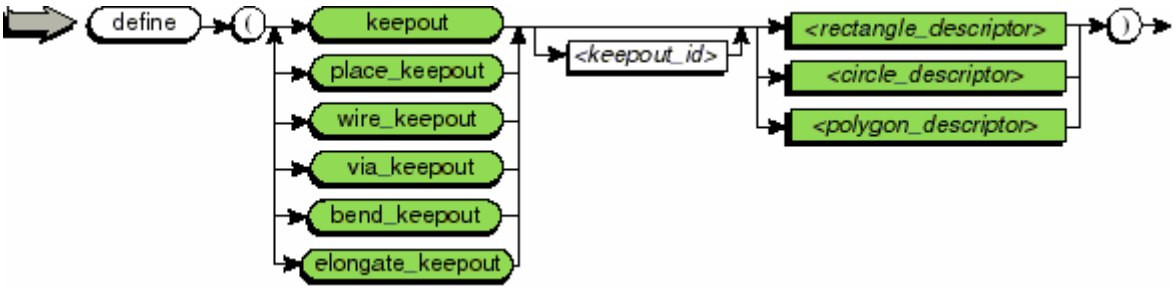
forget command to disband group sets

Command examples

```
define (group g5 (fromto U1-13 U3-10)
   (fromto U3-15 U4-7))
define (group g6 (fromto U1-9 U3-16)
   (fromto U3-14 U4-6))
define (group_set grpset1 g5 g6 (rule (limit_vias 5)))
define (group g7 (fromto U1-12 U3-9)
   (fromto U3-14 U4-6))
define (group_set grpset1 (add_group g7))
```

define keepout

The **define keepout** command defines a keepout area.



keepout

Defines a general keepout area and assigns it a unique name (<*keepout_id*>). A general keepout is an area of the PCB where all routing and placement objects (wires, vias, components, and pins) are prohibited.

place_keepout

A placement keepout is an area of the PCB where components and pins are prohibited. Each keepout area must have a unique name (<*keepout_id*>).

wire_keepout

A wire keepout is an area of the PCB where wires are prohibited. Each keepout area must have a unique name (<*keepout_id*>).

bend_keepout

A wire bend keepout is an area of the PCB where wire bends are prohibited. Each keepout area must have a unique name (<*keepout_id*>).

via_keepout

A via keepout is an area of the PCB where vias are prohibited. Each keepout area must have a unique name (<*keepout_id*>).

elongate_keepout

An elongate keepout is an area of the PCB where wire elongations are prohibited. Each keepout area must have a unique name (<*keepout_id*>).

This command lets you define new keepout areas. A keepout area is an area of the PCB where you prohibit routing or placement. The type of keepout area you specify determines which objects are prohibited.

When you define a keepout area, you specify a keepout type (keepout, placement keepout, wire keepout, via keepout, wire bend keepout, or wire elongation keepout) and describe the shape of the area (rectangle, circle, or polygon) and its location on the PCB. SPECCTRA treats all keepout area shapes as enclosed areas. Prohibited objects are not allowed to touch or cross a keepout area outline.

You can also assign a keepout name (<*keepout_id*>). If you do not assign a name, SPECCTRA assigns one for you. Default keepout names are assigned sequentially beginning with the name *keepout1*.

You can also define keepout areas by drawing them in Draw Keepout mode (see the mode command for details). Use **define keepout** when you want to provide precise X and Y coordinates for each corner of a keepout.

You cannot assign rules to keepout areas you define in SPECCTRA, or change rules assigned to keepout areas in the design file.

Notes

Use the forget command when you want to disband a keepout area. You can use the view or vset command to display or hide keepouts in the SPECCTRA work area.

You can change the shape of a keepout area created with the **define keepout** command or that defined in the structure section of the design file using Add/Edit Polygon mode. You cannot change the shape of a keepout area defined as part of a component image in the library section of the design file.

To change the location of a keepout area created with the **define keepout** command or defined in the structure section of the design file, you must disband the keepout area and redefine it. Use the **forget** command to disband the keepout area. Then use **define keepout** to redefine it in a new location. You cannot change disband a keepout area defined as part of a component image in the library section of the design file.

Command examples

This example defines a rectangular keepout.

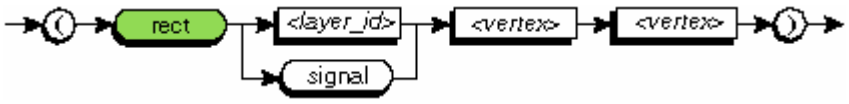
```
define (keepout (rect signal 1.550 4.890 7.630 9.750))
```

This example defines a polygon-shaped placement keepout.

define (place_keepout plc_keep1 (polygon s1 0.1 1.500 4.750 -2.375 4.750 -2.375 3.000 0.250 3.000 0.250 2.000 -2.375 2.000 -2.375 0.500 1.500 0.500 1.500 4.750))

<rectangle_descriptor>

The <*rectangle_descriptor*> defines a rectangular area for a keepout area.



rect

Defines a rectangular area either on a single signal layer or on all signal layers. A <layer_id> is the name of a signal layer defined in the design file.

- For a keepout area, you can use either <layer_id> to identify a specific signal layer or **signal** to identify all signal layers in the PCB.
- For a room, you can use either <*layer_id*> to identify the front or back side, or **signal** to identify both sides, of the PCB.
- For the placement boundary, you must use **signal** to identify all signal layers in the PCB.

<vertex>

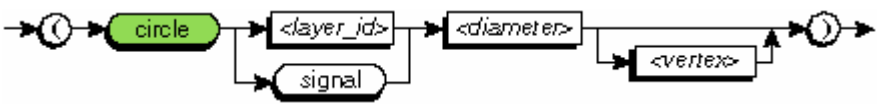
The X and Y coordinates for a point location on the PCB. Depending on how it is used in a command, a vertex can identify an absolute location, a relative location with respect to the PCB origin, a relative location with respect to an object's origin, or a corner of a shape you are describing.

Separate the X and Y coordinates with blank spaces. For example, to specify two vertexes with the coordinates (0, 2) and (3,1), enter 0 2 3 1.

Use **rect** to define a rectangular area on a specific signal layer or on all signal layers in the PCB. You must specify the X and Y coordinates (<*vertex*>) for each of two diagonally opposed corners of the rectangle.

<circle_descriptor>

The <*circle_descriptor*> defines a circular area for a keepout area.



circle

Defines a circular area either on a single signal layer (<layer_id>) or on all signal layers in the PCB (**signal**). A <layer_id> is the name of a signal layer defined in the

design file.

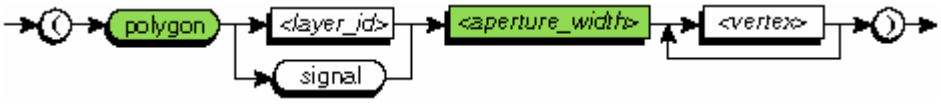
<diameter>

Defines the diameter of a circular area.

Use **circle** to define a circular area on a specific signal layer or on all signal layers in the PCB. You must specify diameter of the circle (<*diameter*>) in the appropriate units for your design. You can also specify the X and Y coordinates (<*vertex*>) for the center of the circle. The default center is the PCB origin defined in the design file.

<polygon_descriptor>

The <*polygon_descriptor*> defines a polygon-shaped area for a keepout area.



Use **polygon** to define a polygon-shaped area on a specific signal layer or on all signal layers in the PCB. You must specify the line thickness (<*aperture_width*>) for the area outline and the X and Y coordinates (<*vertex*>) for each corner of the polygon.

polygon

Defines a polygon-shaped area on a single signal layer or on all signal layers. The <*layer_id*> is the name of a signal layer defined in the design file.

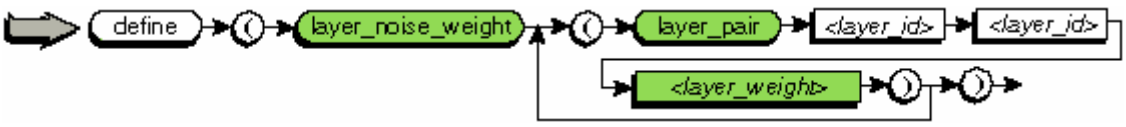
- For a keepout area, you can use either <*layer_id*> to identify a specific signal layer or **signal** to identify all signal layers in the PCB.
- For a room, you can use either <*layer_id*> to identify the front or back side, or **signal** to identify both sides of the PCB.

<aperture_width>

Defines the line thickness for a shape outline such as an area or boundary.

define layer_noise_weight

The **define layer_noise_weight** command creates a table of noise weighting factors that are used by the autorouter when computing parallel noise and tandem noise.



layer_noise_weight

Builds a table of noise weighting factors assigned to specified layer pairs.

layer_pair

Two signal layer names, or a single layer name repeated, that identify the layer(s) <layer_id> for the layer noise weight value assignment.

<layer_weight>

The layer noise weight value to be applied when computing parallel noise or tandem noise between the two layers.

A table entry consists of a pair of layer names and a noise weight factor. You can create a table of entries by entering more than one layer pair - noise weight combination.

Parallel noise and tandem noise

In a parallel noise weight entry, the layer pair repeats one layer name. For example, parallel noise for layer sig1 would be

```
layer_pair sig1 sig1
```

In a tandem noise weight entry, the layer pair consists of different layer names. For example, tandem noise between layers sig1 and sig2 would be

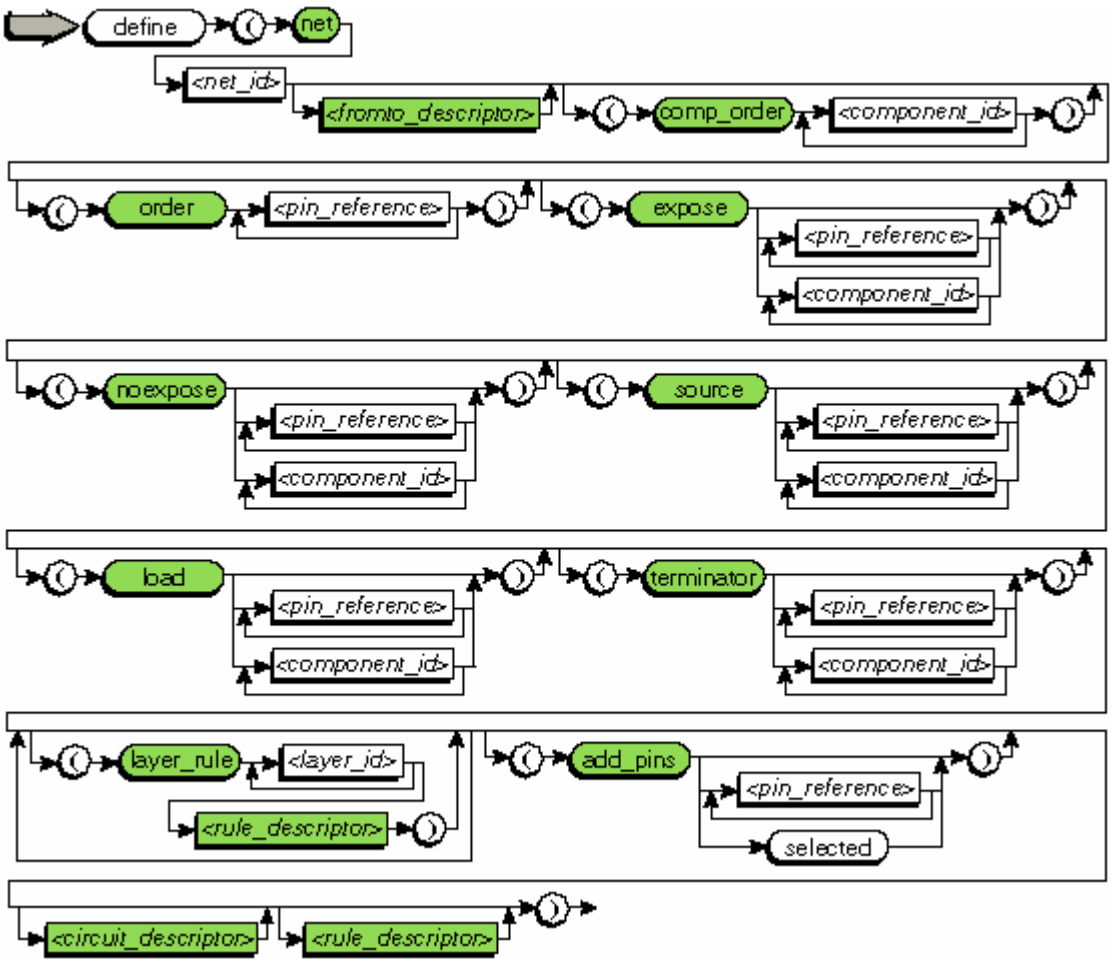
```
layer_pair sig1 sig2
```

Command examples

```
define (layer_noise_weight (layer_pair s1 s1 1.00)
   (layer_pair s2 s2 0.900)
   (layer_pair s5 s5 0.880)
   (layer_pair s5 s6 0.900))
   (layer_pair s1 s2 0.920)
```

define net

The **define net** command applies one or more optional attributes or properties to an entire net, or to specified fromtos in a net. Optionally, it can also apply clearance, timing, shielding, crosstalk, or noise rules to a net or fromtos using the <*circuit_descriptor*> and <*rule_descriptor*>.



net

Specifies a set of pins with the same signal or voltage name to which this command applies one or more optional attributes or properties.

<fromto_descriptor>

Defines a pin-to-pin connection in a net. See the <*fromto_descriptor*> for a complete diagram and description.

comp_order

Orders the pin-to-pin connections of a net by using the component reference designator *<component_id>* only.

order

The pin-to-pin organization of the net. The order in which each <*pin_reference*> is entered in the list determines the pin-to-pin hookup for the net.

expose

An attribute that forces a pin escape to a via on an external layer. You can identify the

pins with <*pin_reference*> or <*component_id*>. The expose attribute applies to through-pins only.

noexpose

An attribute that removes the **expose** attribute for the specified pins so that fanout does not generate vias for those pins.

source

A property assigned to pins for daisy-chain routing. You can identify the pins with <*pin_reference*> or <*component_id*>.

load

A property assigned to pins for daisy-chain routing. You can identify the pins with <*pin_reference*> or <*component_id*>.

terminator

A property assigned to pins for daisy-chain routing. You can identify the pins with <*pin_reference*> or <*component_id*>.

layer_rule

A routing rule that applies to all group_sets on the named layers, unless a higher-precedence rule overrides.

add_pins

Assigns one or more pins of an added component or an existing component to the specified net.

The **define net** command can be used in a do file to assign net rules (such as pin ordering) that are not supported in the host layout system. Although you can use rules in the autorouter that are not available in the layout system, you can't add new circuit elements to a design. For example, you can't add a new net to the design. It is possible, however, to add existing component pins to an existing net with the **add_pins** option.

Note

When you use **define net**, you cannot specify fromtos that form a loop. SPECCTRA issues an error message and stops on the fromto that causes the loop.

See also

circuit and rule commands for complete <*circuit_descriptor*> and <*rule_descriptor*> diagrams and descriptions.

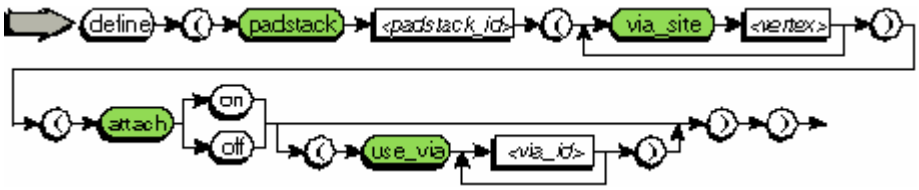
Command examples

define (net sig1 (order U1-1 U2-2 U4-4))

```
define (net sig20 (fromto U10-2 U6-4)
   (fromto U6-4 U20-3
   (rule (width 0.030))) (fromto U20-3 U20-4
   (circuit (use_layer TOP))))
define (net sig1 (order U1-1 U2-2)
   (comp_order U2 U3))
define (net sig2 (expose U7))
```

define padstack

The define padstack command specifies an under pad via location for an SMD pad.



padstack

Specifies an SMD pad stack <*padstack_id*> and one or more under pad via locations for the padstack. <*padstack_id*> must refer to a padstack defined in the design file.

via_site

Specifies a location <*vertex*> relative to the padstack origin where an under pad via can be inserted.

attach

Enables use of (**on**) or removes (**off**) the specified via site(s) for under pad vias.

use_via

Specifies one or more vias for use at the specified via sites.

The under pad via location is specified relative to the SMD pad origin. The location may be under the pad, or offset sufficiently to site the via outside the pad outline.

Note

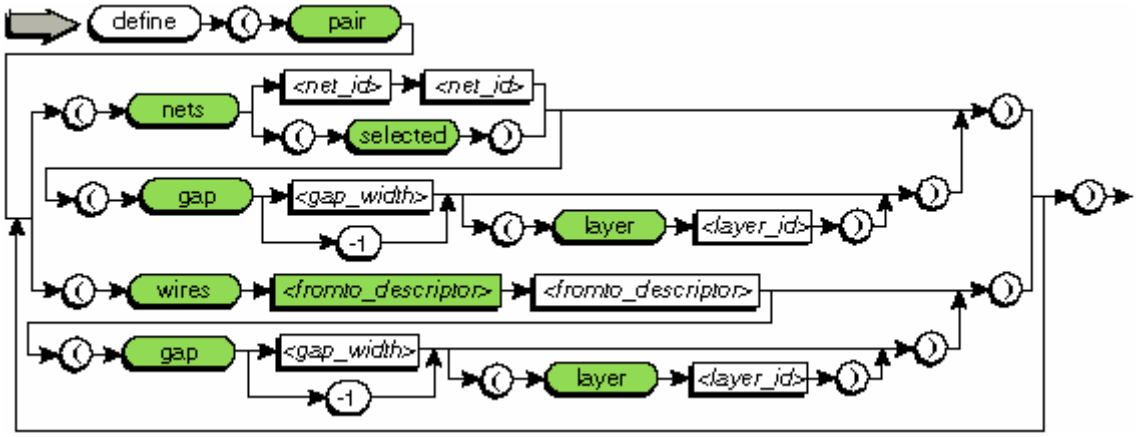
This command must be used in conjunction with the via_at_smd rule. Vias will not be located under pads, or at specified under pad via sites, unless the via_at_smd rule is on.

Command examples

```
define (padstack smd_2 (via_site .07) (attach on))
define (padstack smd_2 (off)
define (padstack smd_2 (via_site .07) (attach on (use_via via_a)))
define (padstack smd_2 (via_site .07) (attach off))
```

define pair

The **define pair** command defines one or more pairs of nets or wires to be routed as a differential pair with the same topology.



pair

Defines one or more pairs of nets or wires to be routed as a differential pair with the same topology.

nets

Specifies a pair of net names (<*net_id*> <*net_id*>) to make up a differential pair.

selected

Applies the pair definition to two, and only two, selected nets.

gap

The edge-to-edge distance between paired wire segments.

<*gap_width*>

The target wire-to-wire spacing between differential pair wires. The autorouter uses a greater wire-wire spacing only when obstructed by an object in the routing path.

-1

Resets **gap** for the differential pair to unspecified.

layer

Applies the specified **gap** option to only the layer specified in <*layer_id*>.

wires

Defines a pair as two fromtos you specify using fromto descriptor syntax.

<fromto_descriptor>

Specifies one of two pin-to-pin connections that make up a differential pair. See the *<fromto_descriptor>* for a complete syntax diagram and description.

When gap is specified for a pair, the autorouter attempts to maintain the gap along the pair's entire length. If you define a pair and set a gap, you can subsequently reset the gap to the default clearance rule by using -1 as the gap value.

Note

By default, the length considered when applying timing rules (length and delay) to pairs is the average length for the pair (calculated by adding the individual lengths of the two wires in the pair, and dividing by two). If you want the nets to be checked independently, use **set average_pair_lengths off** (see the set command for details).

See also

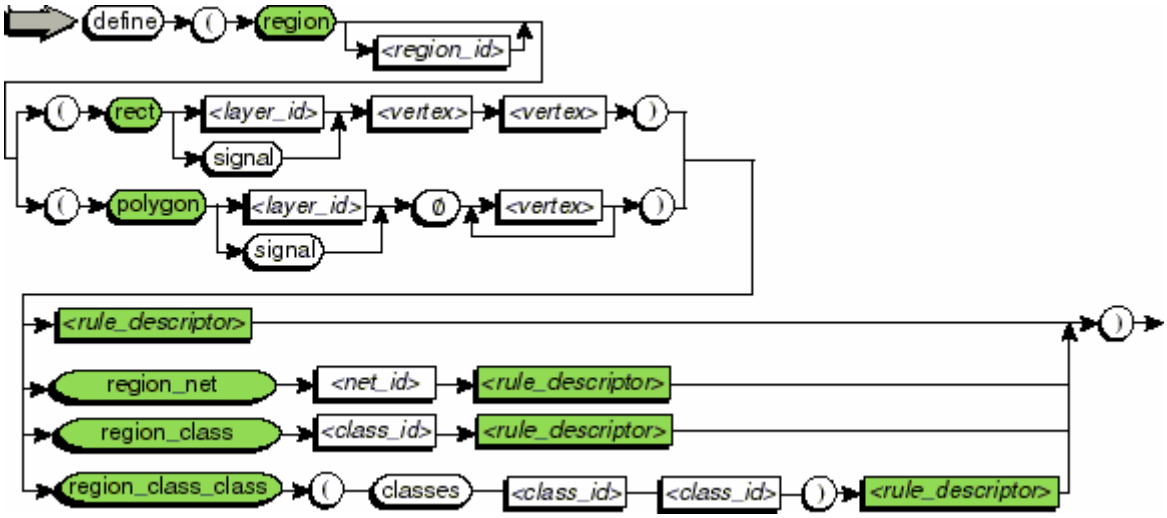
forget pair command to disband differential pairs.

Command examples

define (pair (nets sig16 sig17 (gap 0.005)))
define (pair (nets A?+ A?- (gap 0.008))

define region

The **define region** command creates a rectangular or polygon shaped area within which different width and clearance rules apply.



region

Defines a rectilinear region and assigns one or more rules to it. The optional *<region_id>* assigns a name to the region.

rect

Specifies the rectangular area with two pairs of coordinates <*vertex*> and assigns the region either to a specific layer <*layer_id*> or to all signal layers (**signal**).

polygon

Specifies the polygon-shaped area with three or more pairs of coordinates <*vertex*> and an aperture width of **0**, and assigns the region either to a specific layer <*layer_id*> or to all signal layers (**signal**).

region_net

Assigns clearance or width rules to the specified net (<*net_id*>) within the region. If the region overlaps other regions, region_net rules take precedence over region_class rules and global region rules.

region_class

Assigns clearance or width rules to the specified class (<*class_id*>) within the region. If the region overlaps other regions, region_class rules take precedence over global region rules.

region_class_class

Assigns clearance rules between the specified classes (<*class_id*>) within the region. If the region overlaps other regions, region_class_class rules take precedence over region_net rules, region_class rules, and global region rules.

Use this command to define routing areas where you want different clearances or wire widths to apply than on the rest of the PCB. You can assign rules for the entire region, for a single class or net within the region, or between two classes within the region.

When you define a region, you choose shape (rectangle or polygon), specify its layer and location, and assign the clearance rules, width rules, or both that you want to apply within the region. You can also assign an optional region name.

Region name

The optional <*region_id*> assigns a name to the region. If you omit this option, SPECCTRA automatically assigns a default name. A region name must be unique and can consist of any combination of text characters or symbols, except blank space, parentheses, or semicolon.

You can use the region name in later commands to reference the region. You cannot delete or change the name of an existing region.

Region shape

Use either **rect** to define a rectangular region or **polygon** to define a polygon shaped region. You can define the region for a single layer (<*layer_id* >) or for all signal layers (**signal**). Use <*vertex*> to specify the X and Y coordinates for the diagonally opposed corners of a rectangular region or each corner of a polygon shaped region.

Note: The autorouter recognizes only rectilinear regions. If you define a polygon-shaped region, the autorouter encloses any diagonal side of the region with a rectilinear corner.

Rules and overlapping regions

You define a region when you want different clearance or with rules to apply in the region area than elsewhere on the PCB. The type of region you define depends on precedence level of the rules you assign to the region.

You can assign global region rules to all nets within the region, or you can'

- Use **region_net** to assign clearance and width rules to only the specified net (<*net_id*>).
- Use **region_class** to assign clearance and width rules to only the specified class (<*class_id*>).
- Use **region_class_class** to assign clearance rules between only the two specified classes (**class** <*class_id*> <*class_id*>).

Region rules have the highest precedence in the rule hierarchy. Therefore, within a region, the region clearance and width rules override all other clearance and with rules. See also routing rule hierarchy .

If you define regions that overlap, region_class_class rules take precedence over all other region rules, followed by region_net rules, region_class rules, and global region rules. If two regions of the same precedence level overlap, the rules for the overlapping regions are merged, or if the rules conflict, the rules of the last defined region are used.

Notes

You can also define regions by drawing them in Draw Region mode (see the mode command for details). Use **define region** when you want to provide precise X and Y coordinates for the room outline.

You can use the rule command to assign or change rules in existing regions.

Use the forget command if you want to disband a region.

See also

rule command for complete <*rule_descriptor*> diagrams and descriptions.

Command examples

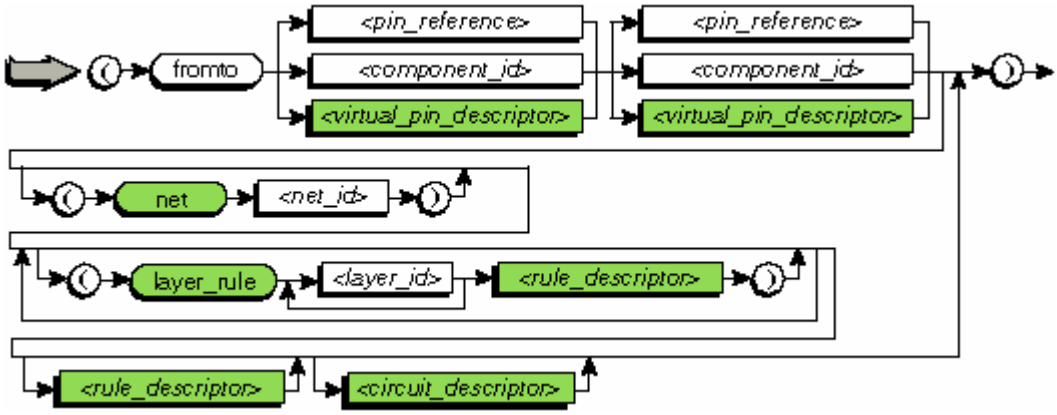
```
define (region region1
  (rect s1 0.975 1 1.75 .6)
  (rule (width 0.015)))

define (region rgn2
  (polygon signal 0
    1.500 4.750 2.375 4.750 2.375 3.000
    0.250 3.000 0.250 4.000
    1.500 4.000 1.500 4.750)
  (region_class class1)
```

(rule (clearance 1.5)))

<fromto_descriptor>

The <*fromto_descriptor*> specifies one or more individual pin-to-pin connections.



<virtual_pin_descriptor>

References a virtual pin, which is a tjunction or via. See the <*virtual_pin_descriptor*> for a complete diagram and description.

net

A set of pins with the same signal or voltage name. The autorouter must connect these pins with wires. Voltage can be assigned to a "power" layer. Each net is defined in the network section of the design file. Every pin of a net is identified by a component reference designator and a physical pin name.

layer_rule

A routing rule that applies to all wires routed on the named layers, unless a higher-precedence rule overrides.

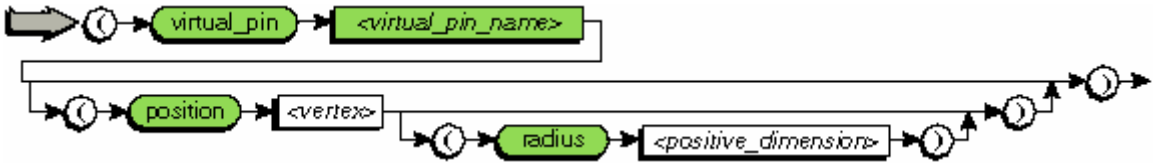
Use the <*fromto_descriptor*> to specify an individual pin-to-pin connection in a net. You can specify a fromto as a pair of pin names, component names, or virtual pins. You can use the net name <*net_id*> to differentiate virtual pins in groups, since virtual pin names are not unique.

See also

circuit and rule commands for complete <*circuit_descriptor*> and <*rule_descriptor*> diagrams and descriptions.

<virtual_pin_descriptor>

The <*virtual_pin_descriptor*> references a virtual pin, which is a tjunction or via.



virtual_pin

References a virtual pin, which is a tjunction or via.

<virtual_pin_name>

Any virtual pin name defined in the design file, or assigned in SPECCTRA. A given virtual pin name can be used in more than one net.

position

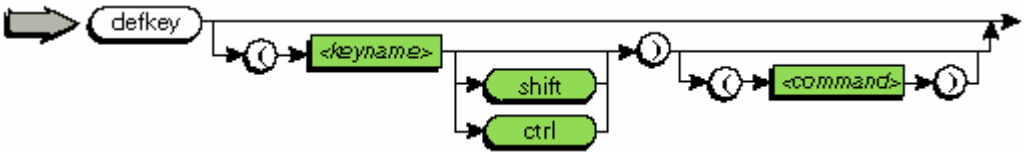
The location of a virtual pin.

radius

The distance a virtual pin can be moved from the vertex to avoid a violation. The default *<positive_dimension>* value is 0.5 inch.

defkey

The **defkey** command displays existing key programming and assigns SPECCTRA commands to keys.



<keyname>

Any alpha-numeric key, including function keys.

Use this option to name the key you want to program or re-program. Depending on the window manager in use, some keys may not be programmable on your system. Generally, the defkey command issues a warning in such cases.

shift

Adds the shift key to <*keyname*> to define a key combination that executes <*command*>.

ctrl

Adds the control key to <*keyname*> to define a key combination that executes <*command*>.

<command>

A SPECCTRA command, typed exactly as if used in a do file or on the command line.

The **defkey** command programs an unused key to execute a SPECCTRA command.

Some keys are predefined in SPECCTRA. You can see a list of keys that have been predefined and keys that you have defined, and the commands they perform, by entering **defkey** without an option.

You can program function keys and standard alphanumeric keys. In general, to execute a defined command, you move the pointer into the work area and press the key(s).

However, programmed alphanumeric keys only execute the assigned command when the keyboard focus is set to the work area . When the focus is set to the command entry area, programmed alphanumeric keys enter a standard keyboard character. You can toggle the keyboard focus by pressing the [Tab] key or by using the set_focus command.

Some function keys you cannot program because they are reserved by the computer hardware, operating system, or window manager you are using.

Function keys that cannot be programmed on UNIX systems are listed for some platforms in the following table.

Key Not User Definable	Platform	Configuration
F1, F4, F10, Shift-Undo	Sun SPARCstation	Solaris with OpenWindows
F1, F4, F7, F9. F10, F11, F12	HP 9000 Series 700	HP-UX with VUE
F1, F4, F10	IBM RISC System/6000	AIX

Function keys that cannot be programmed on Windows 95 or Windows NT systems are reserved for the uses described in the following table.

Key	Used For
F1	Opens help
Shift-F1	Sets point & click help mode
Ctrl+F4	Closes document window
Ctrl+F6	Moves to next document window
F10, Shift+F10, Ctrl+F10	Activates menu without using mouse

You can save your defined keys in a keys file for use in a later session by using the

write keys command or the write environment command.

Note

In SPECCTRA, the [F1] key is predefined to access the SPECCTRA online help. However, in Solaris 5.4 and 5.5 under Open Windows, [F!] is set as the Open Look help key.

If you want to use [F1] to access SPECCTRA help or redefine it to perform some other function, you must first remove or comment the following line in the .xinitrc file in your home directory:

Change

```
xmodmap -e 'keysym F1 = Help'
```

to

```
# xmodmap -e 'keysym F1 = Help'
```

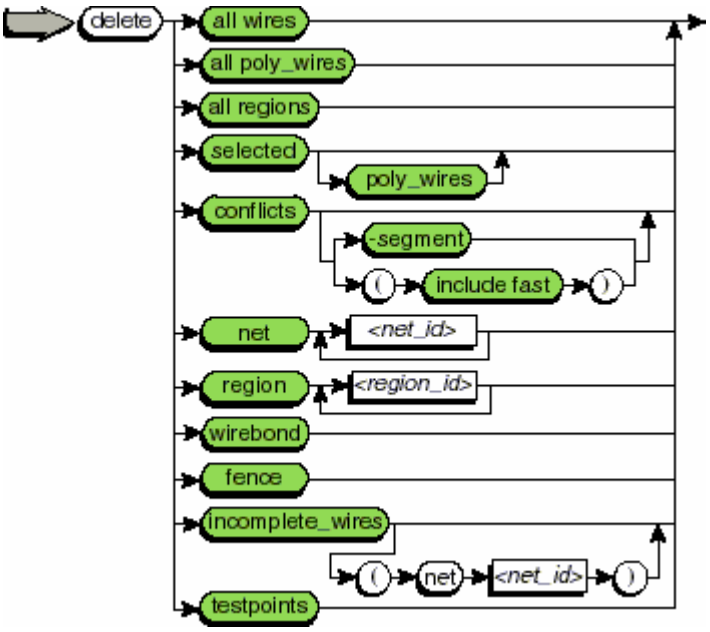
Log out and log back in to apply your edits and redefine the [F1] key.

Command examples

defkey
defkey (f3 ctrl) x
defkey (f5 shift) (undo)

delete

The **delete** command removes objects from the current autorouting environment.



all wires

All routed wires except protected or fixed wires. You are prompted to confirm if the

command is executed interactively.

all poly_wires

All wiring polygons that are not protected. You are prompted to confirm if the command is executed interactively.

all regions

All regions and associated rules, which include shapes defined as regions in the design file structure section, rules associated with component images, and regions defined with the **define** command. You are prompted to confirm that you want to delete regions. After you **delete all regions**, you can use the define region command to create new regions.

If a deleted region had a wire width rule associated with it, the wire width of existing wires will not change until additional route or clean passes are executed. The check command is executed automatically after **delete all regions** to update the conflicts based upon the new rule set.

selected

All routed wires that are selected. For example, if you issue **select component U1**, and then **delete selected**, all routed wires from comp1 to the first terminal point are deleted.

selected poly_wires

All selected poly_wires that are not protected.

conflicts

All routed wires that intersect other routed wires or violate clearance rules are deleted. Starting with the wires that cause the most intersections and clearance violations, the autorouter removes each wire and re-evaluates the violation list.

-segment

The **-segment** option allows the autorouter to eliminate conflicts by removing single segments and creating a guide from one segment to another. The **delete conflicts** command is not recommended when there are a large number of conflicts. Instead, use the filter command to remove conflicts.

include fast

Deletes wires that include violations of high-speed rules, such as length or delay rules.

net

All routed wires for nets identified by <*net_id*>.

region

Areas identified by <*region_id*> and all associated rules.

If a deleted region had a wire width rule associated with it, the width of existing wires does not change until additional route or clean passes are executed.

wirebond

All discrete wires and wirebond pad sites.

fence

All fences.

incomplete_wires

Incomplete wiring in this sense includes:

pin-to-pin connections with a segment missing. Here, “missing” might or might not include guide wires connecting the other segments.

segments that tee into a pin-to-pin connection but end without completing the connection or end at a guide wire.

segments that start at a pin and end without completing the connection (but segments that end at vias are presumed to be fanouts or test points and are *not* deleted).

wires left dangling by the execution of a **delete conflicts -segment** command.

incomplete_wires net

Incomplete wires that are present on the named net are removed. Use this option to remove dangling wire segments on a net. If a net contains incomplete connections as well as other connections that are complete, only the incomplete wire segments are removed.

testpoints

Deletes all testpoints in a design by removing the testpoint attribute from vias and pins. Also removes wiring and vias added with the testpoint rule.

You can modify wiring by removing all wires, wires involved in conflicts, wires in named nets, all incomplete wires, or incomplete wires in named nets. These **delete** options are useful for experimenting with different routing strategies and rules early in an autorouting session.

You can also remove all regions, named regions, fences, or wirebonds with the delete command.

A delete operation is listed in the routing history table of the status report because it can change conflict and unroute information.

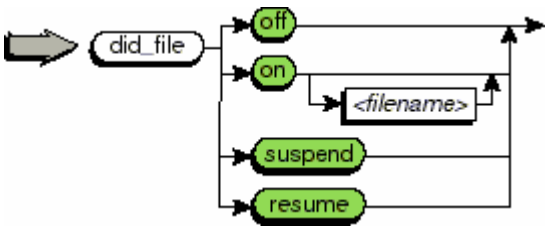
Command examples

delete all wires
delete incomplete_wires (net SYNC1)
delete incomplete_wires

delete poly_wires
delete selected
delete selected poly_wires
delete fence
delete net GND
delete conflicts
delete all regions
delete region region1
delete testpoints

did_file

The **did_file** command controls whether SPECCTRA automatically records commands in a did file.



off

Stops the did file generation, if one is active, and closes the current (active or inactive) file. Subsequent commands are not recorded.

on

Opens a new did file with your specified <*filename*> and closes the current (active or inactive) did file if one is in use. If you do not specify a filename, SPECCTRA generates a new file with a time-stamped filename.

suspend

Suspends the did file generation, if one is active, and changes its status to inactive. Subsequent commands are not recorded until you use either the **resume** option , to resume recording in the currently inactive file, or the **on** option, to close the inactive file and open another file.

resume

Resumes the did file generation in the currently inactive file that you suspended using the **suspend** option. Subsequent commands are appended in the file.

By default, SPECCTRA automatically begins recording commands in a did file when you start a session unless you use the -nodid command line switch. You can use **did_file** to turn off or turn on did file recording, or to specify a different filename, any time during the session. You can also suspend did file recording, and later resume recording in the same did file.

Only one did file can be open at a given time. The status of the current did file is either active or inactive. When you are recording commands, this file is called the active did file. If you suspend recording, the file becomes inactive but remains open. If you later resume recording, the file becomes active again.

- Use **off** to stop recording commands and close the active or inactive did file.
- Use **on** to open a new did file and begin recording commands in the file. If you specify the name (<*filename*>) of an existing file, SPECCTRA overwrites the file. If another did file is currently active or inactive, SPECCTRA closes that file.
- Use **suspend** to temporarily stop recording commands in the active did file. The file remains open but becomes inactive. If you later use **off**, or use **on** and specify a different filename, SPECCTRA closes the inactive file.
- Use **resume** to continue recording commands in the inactive did file where you previously stopped recording commands using **suspend**.

You can use a text editor to edit a did file to create a do file for use in another SPECCTRA session.

The **did_file** command opens, closes, and suspends or resumes command recording in a session did file. This command does not affect the rules did file created with **Edit - Rules Did File** or **File - Write - Rules Did File**. However, you can open a session did file in the rules did file editor and edit the file or record additional commands.

The filename extension that usually identifies a did file is .did, but you can use any filename or extension.

You can choose whether to save or delete the current active or inactive did file when you use the quit command to end the session.

For general information about specifying filenames, see File Naming Conventions.

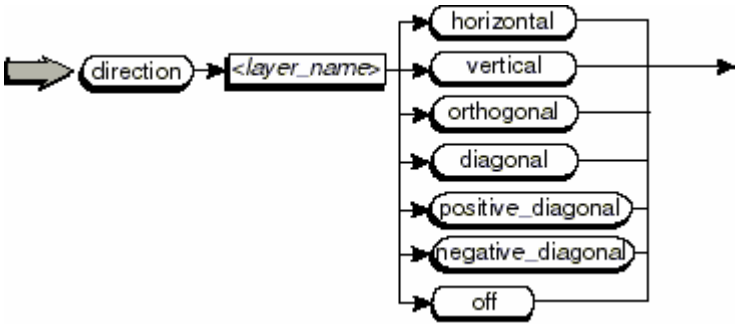
See the *SPECCTRA User Guide* for details about using command line switches to start SPECCTRA.

Command examples

did_file off
did_file on myfile.did
did_file suspend
did_file resume

direction

The **direction** command controls layer routing directions.



The **direction** command sets a preferred routing direction for the specified layer.

Command options are

horizontal sets the preferred routing direction to horizontal and sets a low cost for horizontal routing.

vertical sets the preferred routing direction to vertical and sets a low cost for vertical routing.

orthogonal sets no preference but sets equally low costs for the vertical and horizontal routing directions.

positive_diagonal sets the preferred routing direction to positive diagonal, which is from bottom left to top right and from top right to bottom left, and sets a low cost for positive diagonal routing.

negative_diagonal sets the preferred routing direction to negative diagonal, which is from bottom right to top left and from top left to bottom right, and sets a low cost for negative diagonal routing.

diagonal sets no preference but sets equally low costs for the orthogonal and diagonal routing directions.

off unselects the layer making it unavailable for routing.

The layer routing directions you specify with this command override routing directions set in the design file. If layer routing directions are not set in the design file, the default direction for a layer depends on its position in the structure section. SPECCTRA alternates horizontal and vertical direction assignments. For example, for four signal layers, the default directions are

layer 1, horizontal
layer 2, vertical
layer 3, horizontal
layer 4, vertical

Notes

Diagonal routing is controlled by the set diagonal_mode command and is enabled by default. You should not use **set diagonal_mode off** while a layer direction is set to **diagonal**, **positive_diagonal**, or **negative_diagonal**.

You can also prevent routing on a layer by using the unselect layer command.

Command examples

direction L1 vertical
direction S3 orthogonal
direction S5 positive_diagonal

do

The **do** command reads and executes a do file.

The autorouter reads commands from the specified file. This command file is called a do file. A do file can include any autorouter command.

The **do** command can be executed as follows:

- Keyboard entry, where you enter the **do** command directly in SPECCTRA from the keyboard.
- Menu bar, where you click Execute Do File in the File menu.
- Nested do file, where the autorouter sequentially executes commands in each do file as they are encountered. You can nest up to 20 levels of do files. For example:

Command Sequence	Command Location
do file1	command line entry
grid via 10	do file1 line 1
do file2	do file1 line 2
bus diagonal	do file2 line 1
fanout (pin_type signal)	do file2 line 2
grid via 5	do file1 line 3
route 10	do file1 line 4
write wire mywires	do file1 line 5

When you start the autorouter, you can use the -do switch as a fourth method to execute a do file. If a do file is initiated with the -do switch when you start the autorouter, **do** <*filename*> is the first command executed after the design file is loaded.

If a nested do file is not found in a do file, an alert message displays, but the autorouter continues with the next command in the do file that is running.

Note

You can use the **dofile_auto_repaint** option in the set command to control whether

SPECCTRA repaints the work area after operations performed by the commands in a do file.

See file naming conventions for details about specifying filenames and directory paths.

Command example

do myrules.do

evaluate

The **evaluate** command performs an immediate evaluation of an expression and writes the results (value and type) to the parent shell.

The expression can be a simple variable or a complex computation. The evaluation can be a string, a real number, or an integer.

See the setexpr command for related information.

The internal autorouter variables are defined under <*system_variable*> in the *Design Language Reference*.

Command examples

The following example converts a measurement in centimeters (6.334) to inches by dividing by 2.54.

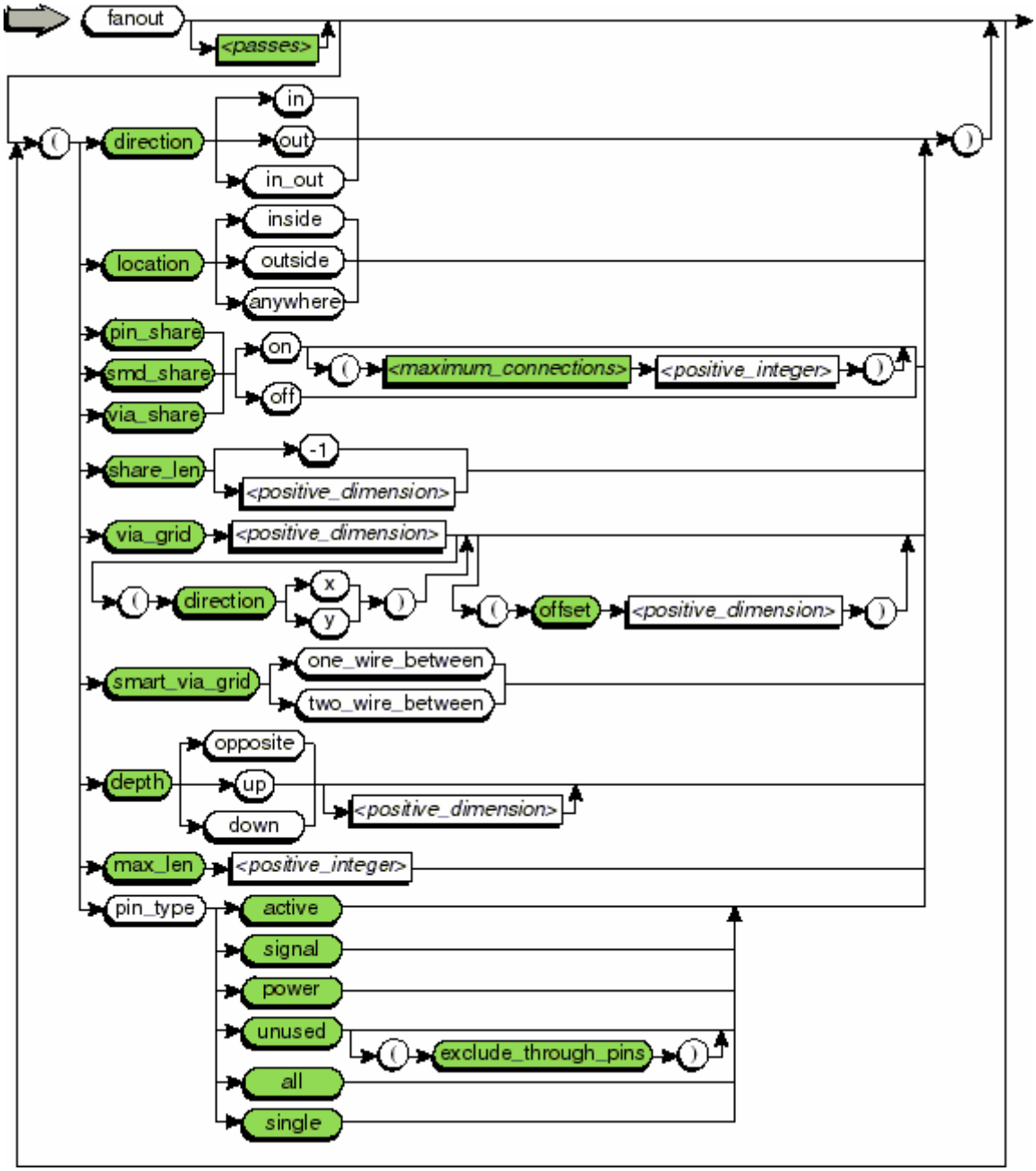
```
evaluate (6.334/2.54)
```

The next example uses a system variable (conflict_wire) to display the current number of conflicts.

```
evaluate (conflict_wire)
```

fanout

The **fanout** command routes short escape wires and vias from SMD pads and through-pins.



<passes>

A positive integer that, when used, must immediately follow the **fanout** keyword to specify the number of rip-up and reroute fanout passes. Conflicts are allowed in the escape wires until the last fanout pass. Five fanout passes are suggested. The default pass value is 1 if not specified.

direction

Directs the autorouter to escape wires and vias inward from the component pins (**in**), outward from the component pins (**out**), or either way (**in_out**). The default **in_out**

option allows the autorouter to escape wires and vias in both directions.

The **direction** you specify also affects how through-pins are escaped when assigned the **expose** property. When the **in_out** option is set for **fanout** (default), exposed pins escape outside the component outline. See the assign_pin command.

location

Directs fanout to escape wires and vias **inside** the component outline, **outside** the component outline, or **anywhere** relative to the component outline.

This option may be used along with the **direction** option to locate vias relative to both the component pins and the component outline. This may be most useful when the component outline extends far beyond the pins.

pin_share

Allows you to control whether the autorouter can escape to through-pins on the same net. The default condition is **pin_share off**, which forces the autorouter to use only vias for escapes. When **pin_share** is **on**, the autorouter escapes to a through-pin on the same net if the cost is lower than the cost to use a via and the pin is within the **max_len** distance.

smd_share

Allows you to control whether the autorouter can connect SMD pins on the same net before escaping to a shared pin or via. The default condition is **smd_share off**, which forces the autorouter to escape SMD pins directly to a pin or via. When **smd_share** is **on**, the autorouter can directly connect SMD pins on the same net if the cost is lower than the cost to use a via and the pin is within the **max_len** distance.

via_share

Allows the autorouter during the fanout operation to share vias between SMD pads on the same net. The default condition, **via_share off**, forces the autorouter to use unique vias for every surface mount pad.

<maximum_connections>

Sets a limit on the number of connections to a pin or via when using the pin_share or via_share option. By default, there is no limit when maximum_connections is not specified.

share_len

Sets the maximum distance that a via or pin can be from a through-pin or via if **pin_share** or **via_share** is on. Vias and pins farther away from these pins will not share a fanout via.

If you use the default (-1), pin sharing can occur with any pin or via within the default distance of 200 mils.

via_grid

Sets a temporary via grid, which is used only during the **fanout** command. If no

fanout via grid is specified, the default is the PCB via grid. This temporary via grid should be a multiple of the PCB via grid.

direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

smart_via_grid

Allows the autorouter to automatically calculate initial via grids that permit one wire or two wires between adjacent vias. If **one_wire_between** is selected, the temporary via grid allows one wire to be routed between adjacent vias. If **two_wire_between** is selected, the temporary via grid allows two wires to be routed between adjacent vias. After **fanout** is completed, the via grid is reset to the original PCB via grid.

If the **via_grid** option is also used, that value is the minimum via grid that is used when computing the smart grid value.

The **preferred** option sets the autorouter to use internal costing to select via sites rather than change the via grid temporarily. This allows fanout to violate the **one_wire_between** or **two_wire_between** specification instead of failing when a suitable via site is not found.

depth

Controls the number of layers a blind or buried via uses during fanout and the direction of the routing. You can set **depth** to the following options:

opposite sets fanout to the opposite side of the design. Pads on the front side fanout toward the back side and pads on the back side fanout toward the front side. Embedded pins, which are pins only on internal layers, fanout to the opposite side from the side to which they are closest.

up sets fanout toward the front side.

down sets fanout toward the back side.

A value of 0 sets no depth limitation.

max_len

Restricts the routed length of the escape wires. The length is measured from a pad's origin to the center of the via.

pin_type

Specifies which types of pins are escaped.

active

All signal pins that interconnect with one or more other pins, and all power pins.

This is the default if no other **pin_type** option is used.

signal

All pins that have signal nets assigned and interconnect with one or more other pins.

power

All pins that have power nets assigned.

unused

All pins, including SMD pads and through-pins, that have no net assigned. Unused pins are collected into a single net called *UNUSED_PINS*.

exclude_through_pin

Excludes unused through-pins from pin escape.

all

All pins on the component including active and unused.

single

All single pin signal nets.

Escape vias are chosen by the autorouter from the available via set and are placed according to the current PCB via grid. Use the **via_grid** option to set a temporary via grid just for the fanout command. Use the **via_grid direction** option to set the grid in only the **x** or **y** direction. Use the **offset** option to offset the first grid point from the 0,0 origin coordinates.

To control fanout direction, use both the **location** option and the **direction** option. Using these options together enables fanout to locate vias relative to both the component pins and the physical component outline. For example where the component outline extends beyond the component pins, and where manufacturing or test requirements demand that fanout vias be accessible, the combination of **direction out** and **location outside** would ensure that fanout vias were directed outward from pins and beyond the physical component outline.

You can select the components you want to escape and designate which pins, and control whether the escape direction is inside or outside the components.

The routing progress indicator monitors and displays the progress of the **fanout** command using a traffic light icon. You can click on the icon to display detailed information in a dialog box.

If you use the **fanout** command without options, it is equivalent to

```
fanout 1 (direction in_out)
   (location anywhere)
   (pin_share off)
   (smd_share off)
   (via_share off)
   (pin_type active)
```

When no components, pins, or nets are selected, all active SMD component pins are escaped. For example, the **fanout** command escapes all SMD pins that are active (have signal or power nets assigned to them). The fanout direction can be both inside and outside of each component's footprint.

Note: "Active" does not include single-pin nets.

You can select the components, nets, pins, and fromtos to escape.

When you...	Then...
Select components	Only the components selected are escaped per the fanout options used.
Select nets	Only SMD pads included in the selected nets are escaped.
Select components and nets	SMD pads that belong to the selected nets *and* the selected components are escaped.
Select pins	Only selected SMD pads are escaped. Unselected pins on the same net are not escaped.
Select fromto	Only SMD pads included in the selected fromto are escaped.

You can use separate **fanout** commands to escape different pin types differently, and you can specify the pin type more than once in a single **fanout** command to handle special situations. For example:

```
fanout (direction in) (pin_type power)
fanout (direction out) (pin_type signal) (pin_type unused)
```

The result of these two commands is that power is routed inside component footprints and signal and unused pins are routed outside component footprints.

The **max_len** rule restricts the routed length of the escape wires. The **max_len** is measured from a pad's origin to the center of the via. Make sure that the **max_len** you specify allows sufficient space so that the wire and via can obey the **smd_via_same_net** clearance rule.

Rules set by using the rule command also affect fanout and are described in the following table.

Rule	Description
smd_to_turn_gap	Sets the minimum distance to the first bend point in a wire from an SMD pad.
smd_via_same_net	Sets the minimum distance from an SMD pad to the first via in the wire.
power_fanout	Controls the routing order between power pins, vias, and bypass capacitors.

Before you use **fanout**, consider the following:

- The **fanout** operation can assist the autorouter on PCBs with four or more signal layers, but is usually not used with two-layer PCBs.
- You can specify a via grid that is used during **fanout**, and reset to the original PCB via grid after fanout is completed.
- You can specify a smart via grid that allows one wire between vias or two wires between vias. This grid is used during **fanout**, and reset to the original PCB via grid after fanout is completed. An additional option to smart_via_grid, **preferred**, sets the autorouter to use internal costing rather than change the via grid. This allows a fanout attempt to violate the one_wire_between or two_wire_between specification rather than fail if a suitable via site cannot be found.
- Rather than protect fanout escape patterns to ensure that all SMD pins have a via for testing, assign test points to signal nets with the testpoint command after routing is complete. The autorouter tries to move existing vias onto the test grid, if you define one, before creating new test point via sites.
- Use the bus diagonal command before **fanout**. The **bus** command executes quickly, and its results can be reviewed before executing further commands.
- If you don't want to fanout all pins of a component, use the select pins or select area pin command to select individual pins. Only selected pins are escaped when you execute **fanout**.
- If you enter **fanout (pin_type all)**, all pins are escaped, including pins without nets attached.
- Single-pin nets are not escaped unless **fanout (pin_type all)** or **fanout (pin_type single)** is used.

Note

With the MicroVia option, fanout behavior changes to provide enhanced support for stacked vias under SMD pads. Click here for a description of this feature.

See also

highlight
protect
smart_route

Command examples

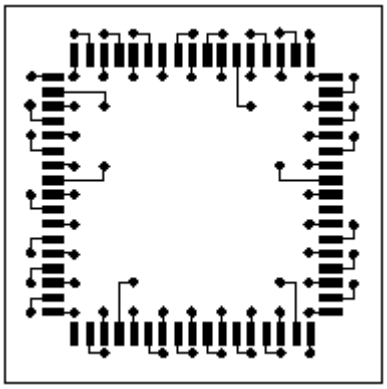
fanout
select component U254
fanout (pin_type all)
fanout (depth opposite 2) (share_len .5)
fanout 5 (pin_type signal) (via_share on (maximum_connections 2))

fanout (smart_via_grid two_wire_between)
fanout (smart_via_grid one_wire_between preferred)
grid via .100 V25
fanout (via_grid .025)

fanout (pin_type unused (exclude_through_pins))

fanout (direction out) (location outside)

An example that shows results of the **fanout (pin_type all)** command is shown below.



Microvia fanout under SMD pads

Blind and buried fanout vias can be created under SMD pads using the fanout command. When the MicroVia option is on, the fanout command works in a way that attempts to allow fanout vias under SMD pads even when the pads are directly opposite each other on either side of the board.

To accomplish this, the fanout command may first move one via off-center from its SMD pad, but still within the pad, to avoid a conflict with a fanout via from the opposite side of the board. This allows each via to achieve its full layer span while remaining under the pad. If this approach fails, then the fanout command may adjust the layer span of one fanout via to avoid a conflict with a fanout via under a pad on the opposite side of the board. Which vias layer span gets adjusted depends on which side of the board is given priority in the fanout command **depth** option.

fence

The **fence** command is used to create one or more route keepin areas or to separate analog and digital signals.

You can define rectangular fence areas to route only the connections that fall completely within that area (hard fence), or to allow analog and digital signals to be routed in separate areas (soft fence). The vertexes are the X, Y coordinates for the opposite corners of the fence area. You can define multiple fences. If fences overlap, the actual keepin area is the union of all the overlapped fences. To remove all fences, use the delete fence command.

Choose Menu Commands in the Help menu for information about using the mouse to draw rectilinear polygonal fences.

Note

You can't have both hard and soft fences in a design. All fences in a design must be either hard or soft.

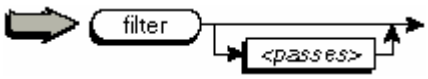
See also set soft_fence for an explanation of hard and soft fences.

Command examples

fence 0.6 1.35 1.0 0.85
fence 1.05 1.38 1.73 0.8

filter

The **filter** command removes final routing conflicts by executing route passes that increase the conflict cost and minimize the number of unconnected wires.



If a few conflicts remain after a large number of route and clean passes are completed, you can use **filter** passes to ensure conflict-free routing with maximum completion. When you initiate the **filter** operation with more than one pass, each pass progressively increases the cost of routing conflicts. During the last filter pass, conflicts are prohibited and any remaining conflicts become unroutes or unconnects. The maximum number of filter passes is five.

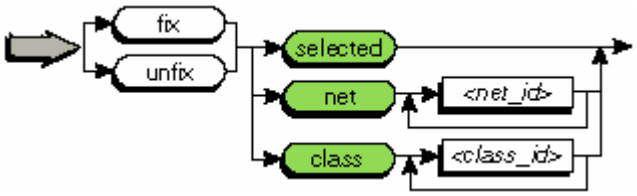
If you issue **filter** without a pass number, SPECCTRA executes a single pass.

Command examples

filter
filter 5

fix/unfix

The **fix** command prevents routing and rerouting of nets.



Note: See Notes and Command examples for additional syntax not shown in this diagram.

selected

Only the nets that are selected are fixed. The entire net is fixed, including any partially routed connections.

net

All terminals and routed wiring for the specified nets.

class

All nets in the specified classes.

This command ensures that selected or specified nets are not altered by any subsequent autorouter operations. Neither the wired nor unwired portions of a fixed net can be modified by the autorouter until an **unfix** command is used to change the net's status. Wires of fixed nets are treated as keepouts and cannot be involved in conflicts.

The **unfix** command restores the normal status of nets that have been fixed with the **fix** command in SPECCTRA. It does not affect wires marked as (type fix) or (type route) in the design file. Wires from the wires file or design file of (type fix) or (type route) can only be changed by editing the design file. Many translators use the absence of (type fix) and (type route) to know which nets to merge back in the CAD system.

Notes

The fix and unfix commands operate only on nets or fromtos. See the protect/unprotect command to control the rerouting of wires.

The fix and unfix commands also operate on groups of fromtos. See the command examples for syntax that is not shown in the diagram.

Command examples

fix selected
unfix selected

fix selected group
unfix selected group

fix group group1
fix group group2 group3

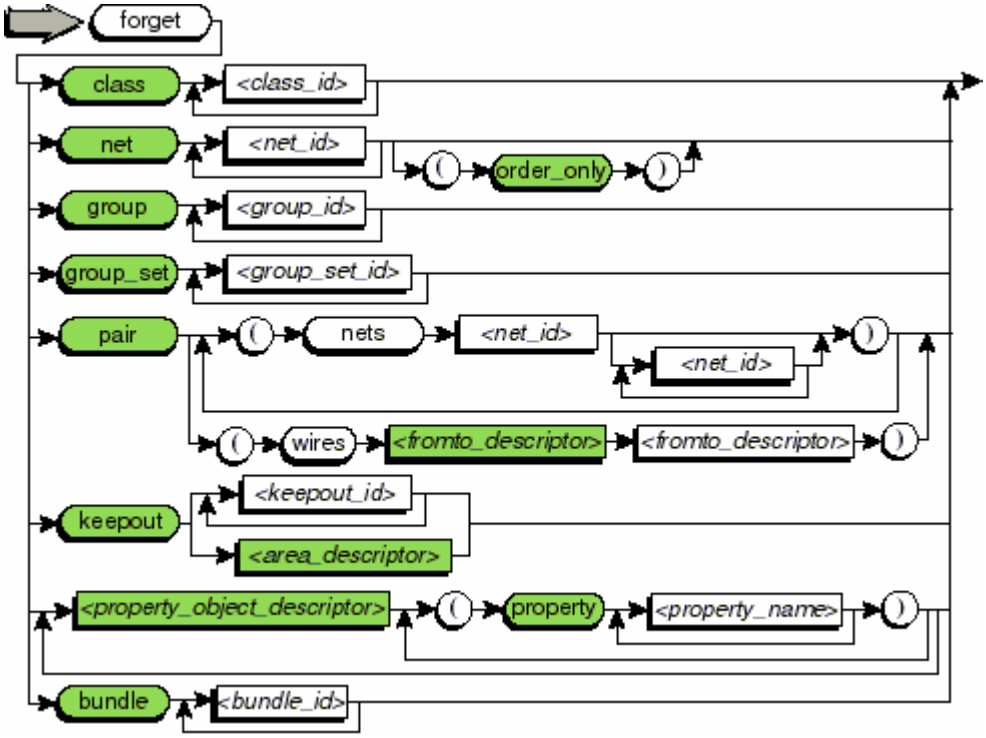
unfix group group1 group2 group3

fix net clk
unfix net clk

fix class critical
unfix class critical

forget

The forget command removes or disbands collections of objects, area objects, and object properties.



class

Disbands the named classes and removes all class and class_class rules from the nets in those classes.

net

Removes all rules assigned to the named nets and forgets any fromto ordering defined for those nets in SPECCTRA.

order_only

Removes only the fromto ordering of a net, leaving all other assigned rules intact.

Deletes fromtos and virtual pins created by the **define net** command.

group

Disbands one or more named groups of fromtos. All group rules (including ordering) that are assigned to the fromtos are no longer in effect.

group_set

Disbands one or more named group sets and removes all assigned clearance, width, and timing rules.

pair

Disbands the pair association between one or more net pairs.

<fromto_descriptor>

Defines a pin-to-pin connection. See the *<fromto_descriptor>* for a complete diagram and description.

keepout

Disbands one or more keepouts. You can disband individual keepouts or all the keepouts within an area you specify by using <*area_descriptor*>. A <*keepout_id*> is the name assigned to an individual keepout you want to disband.

<area_descriptor>

Describes the location or area where you want to disband keepouts. See the *<area_descriptor>* for a complete diagram and description.

<property_object_descriptor>

Specifies a design object for assignment or removal of properties. See the *<property_object_descriptor>* for a complete description.

property

Identifies a property (<*property_name*>) you want to remove.

Each standard or user property consists of a keyword (<*property_name*>) and a value (<*property_value*>). You specify the keyword of the property you want to remove,

Note

You can remove properties assigned in SPECCTRA but not properties assigned in the design file.

bundle

Removes bundle definitions (<*bundle_id*>) created with the **define bundle** command).

The **forget** command can disband classes, groups, group sets, pairs, bundles, and keepout areas, and can remove rules assigned to nets. It can also disband net fromto ordering that you defined in SPECCTRA, and remove properties assigned to design

objects.

SPECCTRA discards any rules assigned to disbanded classes, groups, group sets, or pairs.

- Use **forget class** to disband one or more classes of nets. All class and class_class rules associated with the named classes are also disbanded. After you disband a class, that <*class_id*> can be reused to define a new class.

- Use **forget net** to remove all rules assigned to a net and to forget any fromto ordering that you defined in SPECCTRA. If you want to remove only the fromto ordering of a net without removing rules assigned to it, use the **order_only** option with **forget net**.

  If you use the **order_only** option with **forget net**, any rules associated with fromtos in the net are removed. However, rules associated with the entire net are not removed.

  Note: **forget net** deletes fromtos and virtual pins created by the define net command.

- Use **forget group** to disband one or more groups of fromtos. All group rules (including ordering) that are assigned to the fromtos are no longer in effect. A disbanded <*group_id*> can be reused to define a new group comprising a different combination of fromtos with different rules.

- Use the **forget group_set** command to disband a group set and remove all clearance, width, and timing rules assigned. A disbanded <*group_set_id*> can be reused to define a new group set comprising a different combination of groups with different rules.

- Use **forget pair** to disband the pair association between two nets. You can use either **net** to disband pairs of nets or **wire** to disband pairs of fromtos. When you use **forget pair net**, you can include wildcards in each <*net_id*> to specify multiple pairs.

  If you use **forget pair**, and the paired nets or paired fromtos have routed wires, the pair structure is lost during subsequent route or clean operations.

- Use **forget keepout** to disband one or more keepout areas. You can disband individual keepouts or all the keepouts located within an area you specify.

  - To disband individual keepouts, specify the name (<*keepout_id*>) of each keepout you want to disband.

  - To disband the keepout at a particular location, or all the keepouts within a rectangular area, use *<area_descriptor>* to describe the location or area.

  A disbanded <*keepout_id*> can be reused to define a new keepout in a different area of the PCB. You can use the report command to generate a keepouts report and learn the names of all currently defined keepouts in the PCB.

- Use **forget** *<property_object_descriptor>* to remove properties from design objects.

  - Each <*property_object_descriptor*> identifies an object type (component, component pin, image, image pin, layer, or net) and the names of the objects where you want to remove properties.

- Each **property** keyword identifies a property (<*property_name*>) you want to remove.

  Note: If you remove type properties from components or images, SPECCTRA uses pin counts to identify them as large or small by default.

  You can use the report command to generate a property report of all current standard and user-defined properties. Properties are also listed in the component, image, net, and layer reports. Pin properties are listed in the component and image reports.

- Use **forget bundle** to disband the bundle association between the nets of a bundle. If the bundled nets have routed wires, the bundle structure is lost during subsequent route or clean operations.

Note

You can use the report command to generate a report about net ordering and rules. You can also generate reports of all currently defined classes, groups, group sets, pairs, bundles, keepouts, or properties. Disbanded classes, groups, group sets, or keepouts are not included in their respective reports. They also are not available in the dialog box lists when you define rules and add or remove class, group, or group set members.

See also

define bundle
define class
define group
define group_set
define keepout
define pair

mode

component_property
component_pin_property
image_property
image_pin_property
layer_property
net_property

Command examples

```
forget class thin
forget group g1
forget pair (nets sig16 sig17)
forget group_set grpset1
forget pair (nets A?+ A?-)
forget pair (nets *)
forget keepout keepout_1 keepout_2
```

```
forget keepout
  (area 1.550 4.890 7.630 9.750
  (layer signal))
```

forget component_property U1
  (property my_prop_1)

forget component_property
  U2 U3 (property type height)
  U4 U5 (property height)

forget image_pin_property ic1 p3 p5
  (property prop_2 prop_3)

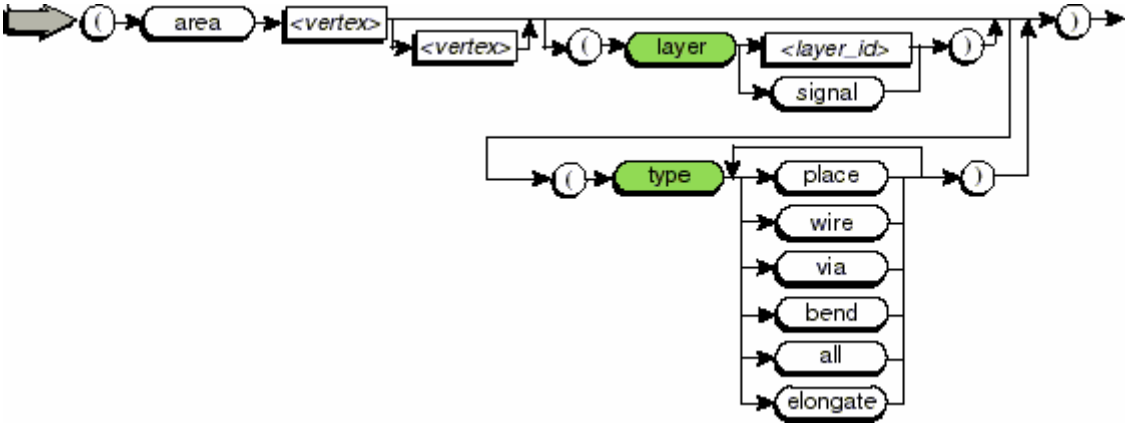
forget image_pin_property
  ic2 p5 p6 (property prop_x)
  p4 p5 (property prop_y)

Note

To identify component or image pins, specify the component name (<*component_id*>) or image name (<*image_id*>) followed by one or more pin names (<*pin_id*>).

<*area_descriptor*>

The <*area_descriptor*> describes the location or area where you want to disband keepouts.



layer

Identifies either a single layer (<*layer_id*>) or all signal layers in the design (**signal**). The <*layer_id*> is the name of a signal layer defined in the design file.

type

Disbands only the types of keepouts you choose within the defined **area**. The choices are

place, which means disband placement keepouts

wire, which means disband wire keepouts

via, which means disband via keepouts

bend, which means disband wire bend keepouts

elongate, which means disband wire elongation keepouts

all, which means disband general keepouts that prohibit all routing and placement objects

The default type is **all**.

You can disband a keepout at a particular location or the keepouts within a rectangular area.

- To disband the keepout at a particular location, specify its X and Y coordinates (<*vertex*>) on the PCB.
- To disband keepouts within a rectangular area, specify the X and Y coordinates (<*vertex*>) for each of two diagonally opposed corners of the rectangle.

By default, SPECCTRA disband all keepouts within the area you describe. You can

- Use the **layer** option and specify a layer name (<*layer_id*>) to disband keepouts only on a single layer.
- Use the **type** option to disband only keepouts of a particular type: keepouts (which prohibit all routing and placement objects), placement keepouts, wire keepouts, via keepouts, wire bend keepouts, or wire elongation keepouts, or general keepouts).

<*property_object_descriptor*>

Identifies an object type and one or more instances of the object. The choices are

component_property <*component_id*>

component_pin_property
[**selected** | <*component_id*> <*pin_id*>]

image_property <*image_id*>

image_pin_property <*image_id*> <*pin_id*>

layer_property <*layer_id*>

net_property <*net_id*>

component_pin_property

Specifies one or more pins of the specified component. A (<*component_id*>) is a component reference designator defined in SPECCTRA or in the design file. A (<*pin_id*>) is the name of a pin on the component. Pin names are assigned to the component's image in the design file or image library file.

See the component_pin_property command for information about assigning properties to component_pins.

component_property

Specifies one or more component_ids. A (<*component_id*>) is a component reference designator defined in SPECCTRA or in the design file.

See the component_property command for information about assigning properties to a component .

image_pin_property

Specifies one or more pins of the specified image. An (<*image_id*>) is the name of an image defined in the design file or in a library file listed in the library section of the design file. A (<*pin_id*>) is a pin name assigned to the image.

See the image_pin_property command for information about assigning properties to an image pin .

image_property

Specifies one or more image_ids. An (<*image_id*>) is an image name defined in the design file or in a library file listed in the library section of the design file.

See the image_property command for information about assigning properties to an image.

layer_property

Specifies one or more layers. A (<*layer_id*>) is the name of a signal layer defined in the design file.

See the layer_property command for more information about assigning layer properties.

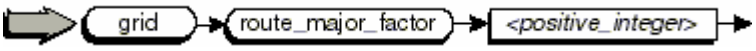
net_property

Specifies one or more nets. A (<*net_id*>) is the name of a signal or power net defined in the design file.

See the net_property command for more information about assigning net properties.

grid route_major_factor

The **grid route_major_factor** command defines a major grid for each wire grid.



When you use **grid route_major_factor**, you specify a value (<*positive_integer*>) to set the major grid for each wire grid. The value specifies the number of minor grid points between major grid points for each wire grid.

For example, if you specify 5 for the major grid and the wire grid is set to .02, the major grid displays every .10 (measurement units).

If you redefine the major grid, SPECCTRA recalculates the major grid points for each wire grid.

Notes

You can use **Define - Color Palette** to change the color of the major grid.

The major grid is a display grid that does not affect routing.

See also

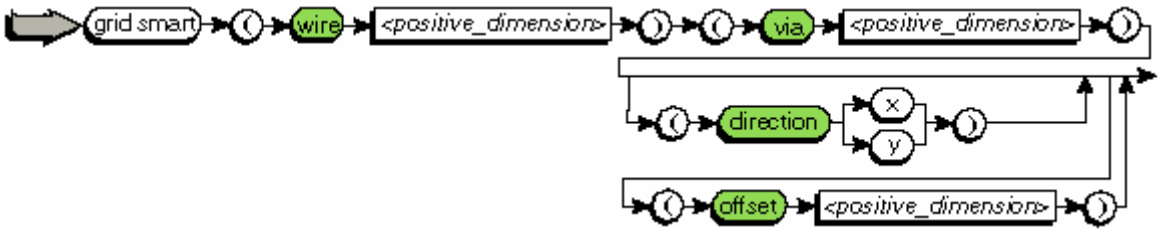
grid wire
view grid

Command examples

grid route_major_factor 5

grid smart

The **grid smart** command sets minimum via and wire grids, and calculates an initial via grid, which is used until the autorouter completes three route passes or the completion rate is 50%. The via grid is then reduced to the minimum value for all remaining routing passes.



wire

Specifies a minimum wire grid.

via

Specifies a minimum via grid.

direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

You can use the **grid smart** command instead of grid wire and grid via commands. The **grid smart** command defines minimum via and wire grid.

The wire grid is set to the value you enter and remains in effect for the rest of the autorouting session, unless you override it with a **grid wire** command. The via grid is set differently. SPECCTRA calculates the initial via grid by using the formula:

via grid = via diameter + 2 x (wire width + wire_via clearance) + wire_wire clearance

Use the **direction** option to set the grid value for only the **x** or **y** direction. Use the **offset** option to offset the first grid point from the 0,0 origin coordinates.

If more than one through-via is available, the autorouter uses the smallest via diameter for the initial via grid calculation. The grid is adjusted upward if necessary so that it is an even multiple of the wire grid.

After routing pass three or when routing is 50% complete, the via grid is set to the value entered. This grid is used for all subsequent routing passes unless you override it with a **grid via** command.

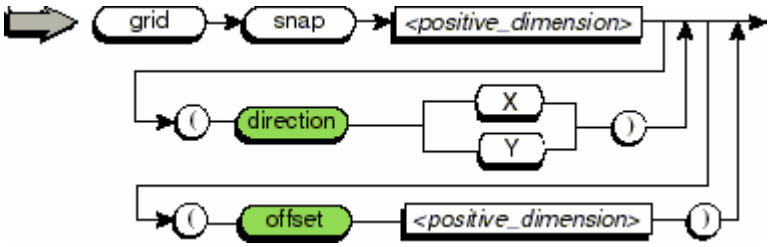
The purpose of **grid smart** is to route initial passes with a larger via grid to avoid via barriers and distribute vias. The smaller via grid prevents via-starving the autorouter during later routing passes. The **grid smart** command allows two wires to route between vias for a few routing passes, and then changes to one wire between vias for better convergence.

Command example

grid smart (wire 0.001) (via 0.025)
grid smart (wire 0.05) (via 0.15) (offset 0.025)

grid snap

The **grid snap** command defines the pointer snap grid points for interactive editing of an area such as a room, keepout, or boundary.



direction

Specifies a grid spacing value in the X direction (**x**) or the Y direction (**y**). If **direction** is not set, the grid value (<*positive_integer*>) specifies a uniform grid in the X and Y directions.

offset

Specifies the grid offset value (<*positive_dimension*>) for the offset of the first grid point from the grid origin (0,0). If you use **offset** with the **direction** option, the offset value applies only to the specified **x** or **y** grid coordinate.

Use this command to set a snap grid that controls pointer movement in the interactive placement and routing modes (see the mode command for details about these modes).

For instance, you can use the snap grid to control pointer movement when you draw areas such as regions, fences, rulers, keepout areas, or rooms. You can also use it when you move objects, edit wires, or add, edit, or cut polygons. The snap grid is not used during any automatic placement or routing operation.

You can specify just a value (<*positive_integer*>) to set a uniform grid in both the X and Y directions, or you can use the **direction** option to set the grid spacing for either the X direction or the Y direction only. Use the **offset** option when you want to offset the first grid point from the grid origin.

The default **grid snap** value is -1, which means no snap grid is used. If you set the snap grid to a value greater than 0, the pointer snaps to the closest grid point as you move it within the work area.

Notes

If you define a manufacturing grid, the X direction and Y direction values for the snap grid must be multiples of the manufacturing grid direction values.

If you also define wire, via, or placement grids, pointer movement is controlled by the wire or via grids (during interactive routing operations) and the placement grid (during interactive placement operations) instead of by the snap grid.

See also

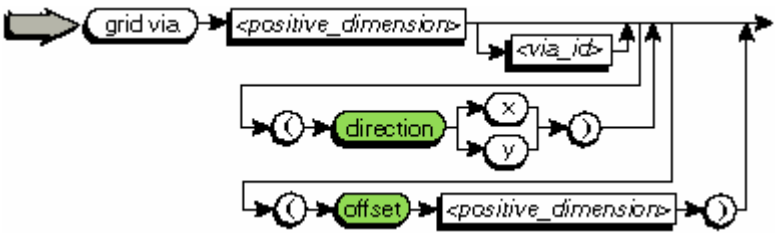
set show_snap_grid_cursor

Command examples

grid snap 0.1

grid via

The **grid via** command defines a via grid.



direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

When you use the **grid via** command, an X, Y via grid is set to <*positive_dimension*> in the current measurement units. The grid is uniform unless you specify a **direction** option. Subsequent autorouting locates any new or rerouted vias on the specified grid. Vias not involved in rip-up and reroute operations remain unchanged. If you include a <*via_id*>, the grid applies only to subsequent use of that via.

Use the **direction** option to set the grid value for only the **x** or **y** direction. Use the

offset option to offset the first grid point from the 0,0 origin coordinates.

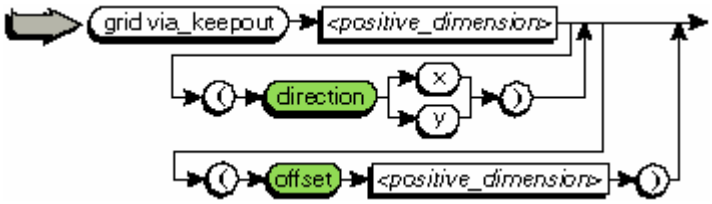
The **grid via** command overrides the via grid computed by the grid smart command.

Command examples

unit mil
grid via 100 V_40
grid via 100 (offset -0.05)

grid via_keepout

The grid via_keepout command controls whether the autorouter can use certain via grid positions.



direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

The autorouter is prohibited from placing vias on grid positions indicated by the <*positive_dimension*> value. The <*positive_dimension*> value identifies X and Y via keepout positions that are referenced to the absolute 0,0 coordinates of your design. A value of 0 removes the grid via keepout.

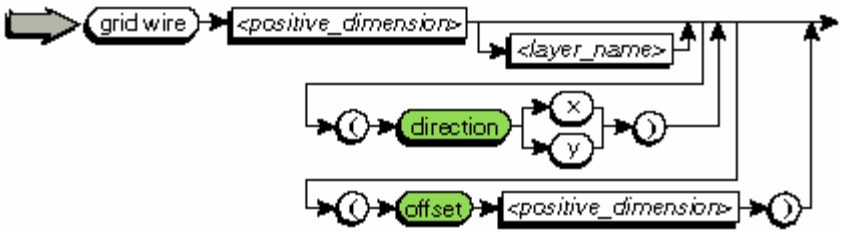
Use the **direction** option to set the grid value for only the **x** or **y** direction. Use the **offset** option to offset the first grid point from the 0,0 origin coordinates.

Command examples

grid via_keepout 100
grid via_keepout 0
grid via_keepout 0.5 (direction x)

grid wire

The **grid wire** command defines a routing grid.



direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

When you use **grid wire**, an X, Y wire grid is set to <*positive_dimension*> in the current measurement units. The grid is uniform unless you specify a **direction** option. New routes or rerouted wires use the specified grid, except when entering or exiting off-grid pins. Existing wires that don't require rerouting are not changed. If you include a routing layer name with the command, the grid applies only to that layer.

Use the **direction** option to set the grid value for only the **x** or **y** direction. Use the **offset** option to offset the first grid point from the 0,0 origin coordinates.

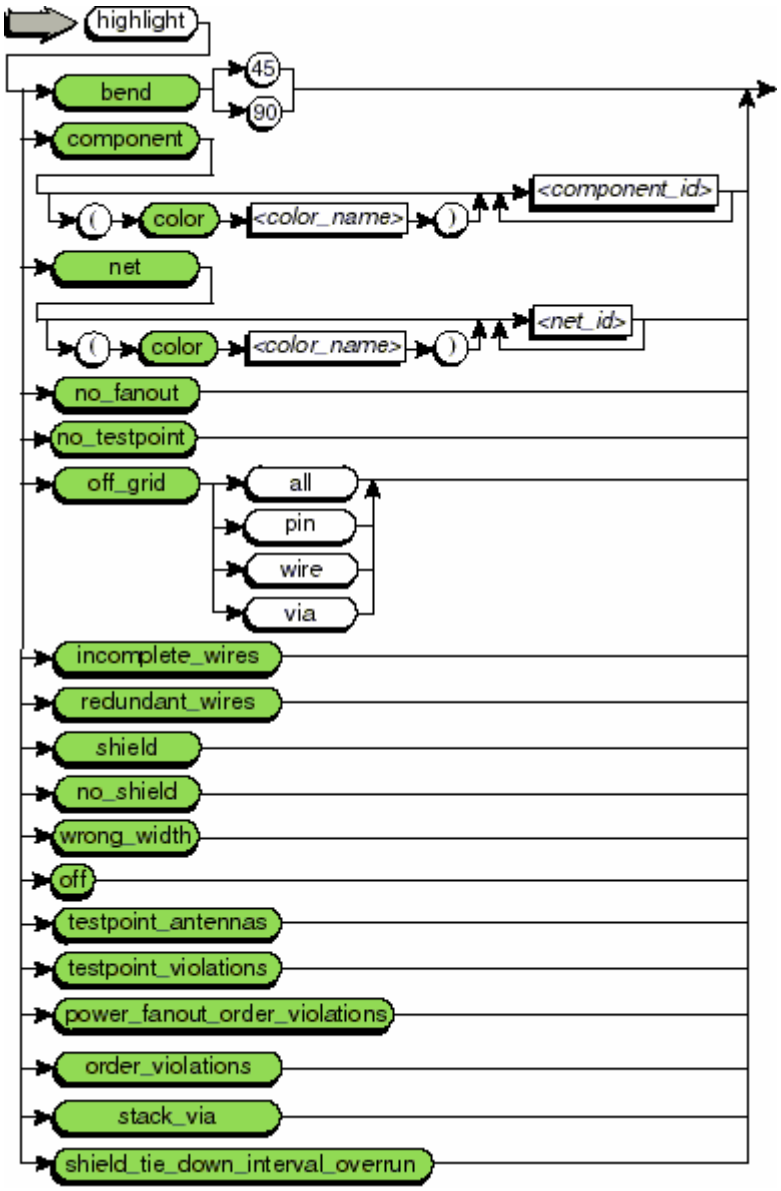
The **grid wire** command overrides the wire grid set by the grid smart command.

Command examples

grid wire 25
grid wire 20 Layer1
grid wire 25 sig1 (direction y)

highlight

The **highlight** command graphically emphasizes certain design objects or routing conditions for easy identification.



bend

All 90 or 45 degree wiring bends.

component

All pins of the named component.

color *<color_name>*

Specifies a highlight color. <*color_name*> can be any color currently defined in the color map.

net

The entire guide of named nets.

no_fanout

All SMD signal pads without escape wires and vias after the last fanout command is executed. If **fanout (pin_type power)** is executed, only power nets that don't contain a fanout are highlighted with **highlight no_fanout**.

no_testpoint

All nets without a testpoint.

off_grid

Pins, wires, or vias that are off-grid and connected to a net. The **all** option specifies all of the shapes listed. Unused pins are not highlighted.

incomplete_wires

Incomplete wiring in this sense includes:

pin-to-pin connections with a segment missing. Here, "missing" might or might not include guide wires connecting the other segments.

segments that tee into a pin-to-pin connection but end without completing the connection or end at a guide wire.

segments that start at a pin and end without completing the connection (but segments that end at vias are presumed to be fanouts or test points and are *not* deleted).

wires left dangling by the execution of a **delete conflicts -segment** command.

redundant_wires

Extra wire segments and vias on nets.

shield

All shielded wires, including the GND shield and tie-in vias.

no_shield

All wires that are supposed to be shielded but were not shielded because the autorouter could not find space for the shield. A wire is not assigned a shield if it is shorter than the min_shield value and it is not highlighted.

wrong_width

Highlights wires whose width has not been changed after executing the change_width_by_rule command.

off

Turns off all highlighting.

testpoint_antennas

All test point antennas. A test point antenna is defined as a test point and associated

wiring connected to a net by a single wire.

testpoint_violations

All testpoints that violate current testpoint rules. Test points with the following violations are highlighted

- the testpoint is on the wrong side of the design
- the testpoint is not on the proper testpoint grid
- there is a testpoint antenna when the rules disallow it
- the wrong type of via was inserted
- antenna is allowed but exceeds the maximum length

power_fanout_order_violations

All connections that violate a **power_fanout** rule. **Power_fanout** rules control the order of fanout connections between power pins, vias, and decoupling capacitors.

order_violation

All out of order routed net connections. Any wire endpoints that constitute a violation are highlighted. The endpoints could be pins, vias with 3 or more connections, or tjunctions. All wiring segments in an order violating fromto are highlighted.

Order violations cannot be highlighted until rule checking has been performed with **order** turned on in the **check** or **setup_check** command.

stack_via

All vias that partially or completely overlap vias behind them on other layers.

shield_tie_down_interval_overrun

All violations of the **shield_tie_down_interval** rule, which sets the maximum distance permitted between stub wires that connect a shield to the ground plane.

Use **highlight** to visually locate objects or conditions in the PCB layout. For routing, you can highlight components, signal nets, and various routing conditions such as wire bends or incomplete wires.

When you highlight a component or signal net, SPECCTRA also highlights the pins, wires, and guides on signal nets connected to the component. When you highlight a signal net, SPECCTRA also highlights the pins, wires, and guides connected to the net.

Objects and routing conditions are emphasized by coloring them with the highlight color (in the default color map). You can change the highlight color by using the color palette (**View - Color Palette**).

Optionally, you can specify a highlight color when highlighting nets or components by name. The optional highlight color remains in effect until highlighting is turned off. The net highlight color (default or optional) overrides the highlight color for pins of that net on currently highlighted components. That is, a pin on a highlighted component is redrawn in the color of its highlighted net.

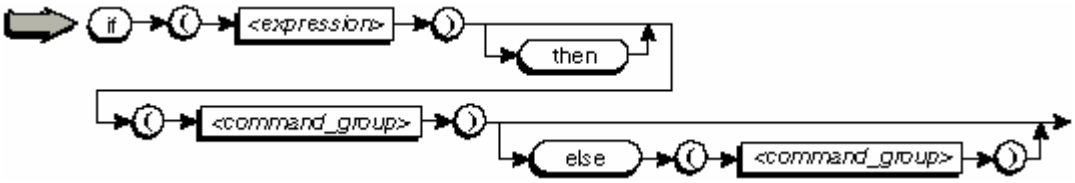
Highlighting does not affect the routing or placement process.

Command examples

highlight bend 90
highlight net +5V
highlight net (color yellow) +5V
highlight no_testpoint
highlight shield

if

The **if** command executes one of two groups of autorouter commands based on evaluating an expression.



The **if** command evaluates <*expression*>. If the value of <*expression*> is not zero, the first <*command_group*> is executed. If the value of <*expression*> is zero, the command group following the **else** keyword is executed. The **else** construct is optional. The purpose of the **if** command is to allow alternative actions during the autorouting process. The <*expression*> can include system variables, which are defined under <*system_variable*> in the *Design Language Reference*. Operators (such as & and !) are defined under the <*numeric_binary_operator*> descriptor in the *Design Language Reference*.

Command examples

The first example initiates 25 route passes, then 2 clean passes. Next, the number of wiring conflicts (conflict_wire) is checked. If there are fewer than five wiring conflicts, two additional clean passes are executed, otherwise 50 additional route passes and 4 additional clean passes are executed. After the **if** block, the autorouter executes the report status command.

```
route 25
clean 2
if (conflict_wire < 5)
   then (clean 2)
   else (route 50 16; clean 4)
report status route.sts
```

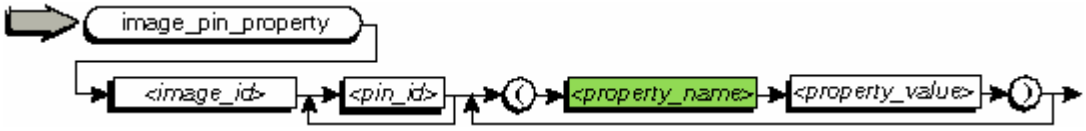
For general information about generating reports, see the Report Commands.

The next example uses system variables to determine whether the design includes SMD components, and whether the top, bottom, or both layers are unselected. If true, five fanout passes are executed. Otherwise autorouting proceeds by starting with the **route 25** command.

```
if (smd_pins && ! (top_layer_sel && bottom_layer_sel))
   then (fanout 5)
    route 25
    if (conflict_wire < 5)
       then (clean 2)
       else (route 50 16; clean 4)
```

image_pin_property

The **image_pin_property** command assigns properties to image pins.



<property_name>

A keyword that identifies a standard property or a user property. Each property you assign must consist of a keyword (<*property_name*>) and a value (<*property_value*>). The value might be another keyword, a number, or a character string depending on what the property requires.

This command lets you assign both standard properties and user properties to one or more pins on an image. You must specify the image name (<*image_id*>) and each pin name (<*pin_id*>).

A property consists of the keyword <*property_name*> that identifies the property, and a value <*property_value*>. Property values can be numbers, keywords, or character strings depending on the property.

The standard properties for image pins include

force_to_terminal_point <*property_value*>
exit_direction <*property_value*>

Properties can be assigned in SPECCTRA or in the design file, but a property assigned to a pin in the design file cannot be changed or removed in SPECCTRA. Image pin properties apply to all instances of the image, but a component pin property value assigned to a specific component pin takes precedence over the value assigned to that property for the image pin.

You can use the report command to generate a property report that contains the current values of properties assigned to all image pins in the design.

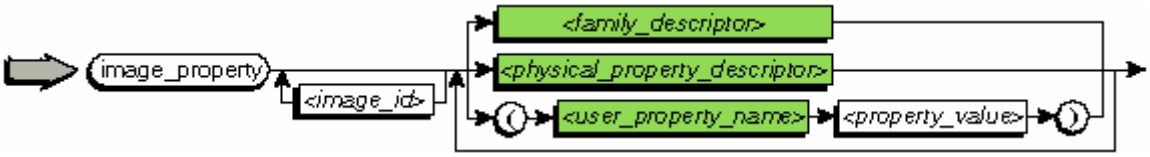
Command examples

image_pin_property C81 2 (uprop_1 0.02)

image_pin_property I6301 3 5 9 (uprop_2 xyz)

image_property

The **image_property** command assigns physical, family, and user properties to images.



This command lets you assign both standard properties and user properties to one or more images. The standard image properties consist of physical properties and family names. Physical properties consist of type, height, and power dissipation.

In general, a property consists of the keyword (<*property_name*>) that identifies the property, and a value (<*property_value*>). Property values can be numbers, keywords, or character strings depending on the property. See image properties for a list of properties you can assign to images.

You can either select the images before using this command or specify the name (<*image_id*>) of each image. If you do not specify image names, SPECCTRA assigns the properties to all selected images.

Properties can be assigned in SPECCTRA or in the design file. Image properties apply to all instances of an image, but a component property value assigned to a specific component instance takes precedence over the value assigned to that property for the component's image.

The standard image properties consist of physical and family properties.

- The physical properties identify an image's type, maximum height, and maximum power dissipation.
- The family properties assign or remove image family names used to assign family-to-family pad and body edge spacing rules.

You can use the report command to generate a property report that contains the current values of properties assigned to all images in the design. You can also generate a total power dissipation report for the PCB and an image family report of all image families.

Note

If you assign or remove physical or family properties on images, SPECCTRA does not record these changes when you use the write command to save a placement file or a session file. Physical properties assigned to individual component instances (using **component_property**), or removed from components, are recorded in these files.

See also

autodiscrete
autorotate

define room
initplace
interchange
place_rule
room_rule
select component
select family
select image
unplace

Command examples

The following examples assign properties to the named images.

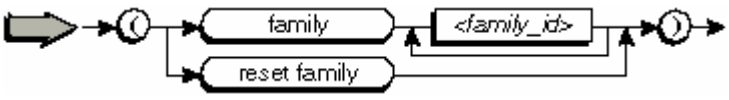
```
image_property C0805 (type capacitor) (height 0.0280)
image_property plcc_20 plcc_28 (height 0.1800 -.1200)
image_property SOIC14 (power_dissipation 500)
```

The following examples assign properties to all selected images.

```
select image IC62 IC63
image_property (height -1 0.051)
image_property (family fam_1)
```

<family_descriptor>

Use <*family_descriptor*> to define a family of images.



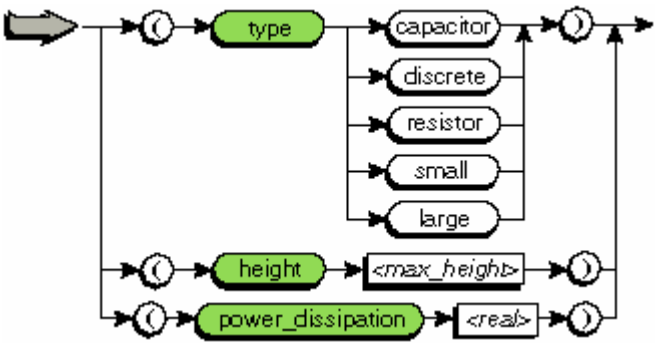
The **family** property is a label you assign that identifies an image as a member of an image family. you can assign the same image to more than one family, and a family can contain one or more images.

After images to a family, you can use place_rule to assign pad edge and body edge spacing rules between images in the family and images in other families.

Use **reset family** when you want to remove images from a family.

<physical_property_descriptor>

Use <*physical_property_descriptor*> to assign type, height, and power dissipation properties to components or images.



type

Controls which small components are included for processing in the current automatic placement operation. A small component is a component with three pins or less that has not been assigned the large type property. The choices are

capacitor, which includes only small capacitors (small components assigned the capacitor type property, and small components with all pins connected to power nets and not assigned the resistor or discrete type property).

discrete, which includes only small discretes (small components assigned the discrete type property).

resistor, which includes only small resistors (small components assigned the resistor type property).

small, which includes all small components.

The default is **small**.

height

Assigns maximum and minimum component height constraints for a room. A value of -1 for <*max_height*> or <*min_height*> means that height constraint is undefined. The defaults are both -1.

power_dissipation

Assigns a maximum power dissipation value for total dissipation of all components in the room. A value of -1 means the power dissipation constraint is undefined. The default is -1.

The physical properties you can assign to a component or image consist of one or two types (**type**), maximum height (**height**), and maximum power dissipation (**power_dissipation**). You can assign or change any or all of these properties in the same command.

- Use **type** when you want to classify components for placement rules or for exclusive processing in automatic placement operations.
- Use **height** when you plan to constrain the minimum or maximum height of components permitted in a room.
- Use **power_dissipation** when you plan to constrain the maximum total power dissipation permitted in a room.

SPECCTRA recognizes the following component and image types:

- Large
- Small
- Capacitor
- Resistor
- Discrete

By default, a large component or image has more than three pins, and a small component has three pins or less. The large and small types are mutually exclusive. Assigning one of them removes the other. You can assign the large type to a component or image with three pins or less, but you cannot assign the small type to a component or image with more than three pins.

You can assign the capacitor, resistor, or discrete type to any small or large component or image. These types are mutually exclusive. Assigning one of them to a component or image removes either of the others.

A capacitor in SPECCTRA is defined as a decoupling (bypass) capacitor. If a component with three or fewer pins, all connected to power nets, has not been assigned the large, resistor, or discrete type, SPECCTRA automatically treats the component as a capacitor.

SPECCTRA distinguishes between large and small components for processing in automatic placement operations. You can also specify small capacitors, resistors, or discretes for exclusive processing. Large capacitors, resistors, or discretes must be processed with other large components.

You can assign separate image set placement rules for each type on the PCB or within a room. Capacitor, resistor, or discrete type rules take precedence over large or small type rules. See place_rule for details.

Note

See the define room and room_rule commands for details about setting placement constraints for rooms.

If you assigned jumper heights to jumpers in the design file and you want to route jumpers beneath components, you must assign to each component (or its image) a **height** property with a value that is greater than any jumper height assigned to jumpers in the design file.

<user_property_name>

A keyword that identifies a user property. Each property you assign must consist of a keyword (<*property_name*>) and a value (<*property_value*>). The value might be another keyword, a number, or a character string depending on what the property requires.

A user property is treated as a label in SPECCTRA, but can have functional meaning to the host layout system or a third party tool.

layer_property

The layer_property command assigns properties to layers.

<property_name>

A keyword that identifies a standard property or a user property. Each property you assign must consist of a keyword (<*property_name*>) and a value (<*property_value*>). The value might be another keyword, a number, or a character string depending on what the property requires.

This command lets you assign both standard properties and user properties to one or more signal or power layers. You must specify the layer name (<*layer_id*>) for each layer.

A property consists of the keyword (<*property_name*>) that identifies the property, and a value (<*property_value*>). Property values can be numbers, keywords, or character strings depending on the property.

There currently are no standard properties available for layers.

Properties can be assigned in SPECCTRA or in the design file, but a property assigned to a layer in the design file cannot be changed or removed in SPECCTRA.

You can use the report command to generate a property report that contains the current values of properties assigned to all layers in the design.

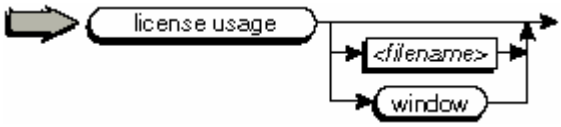
Command examples

layer_property s1 (uprop_1 0.02)

layer_property s1 s2 (uprop_2 xyz)

license usage

The **license usage** command is used to check available and used licenses.



The **license usage** command creates a License Usage Report that lists licenses available and licenses used. You can write the report to a file if you include a filename instead of the **window** keyword. If you issue the command with neither a filename nor the **window** keyword, the report is written to license.rpt in your current directory.

The following list shows the valid license names.

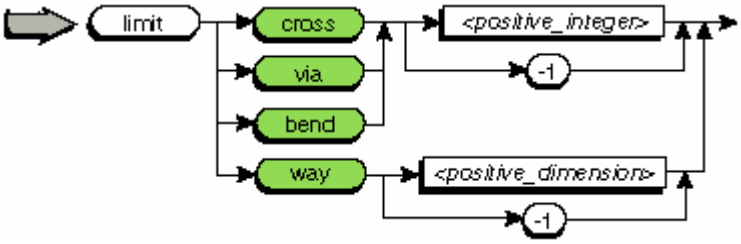
Command example

license usage window

For general information about generating reports, see the Report Commands.

limit

The **limit** command sets absolute controls that apply to each connection for the number of intersecting wires, number of vias, number of bends, and the maximum distance of wrong-way routes.



cross

The maximum number of crossing conflicts allowed when routing a connection.

via

The maximum number of vias that can be used to route a connection.

bend

The maximum number of bend points that can be used to route a connection.

way

The maximum wrong-way distance allowed for a connection.

The **limit** command allows you to specify global routing controls that apply to all pin-to-pin connections.

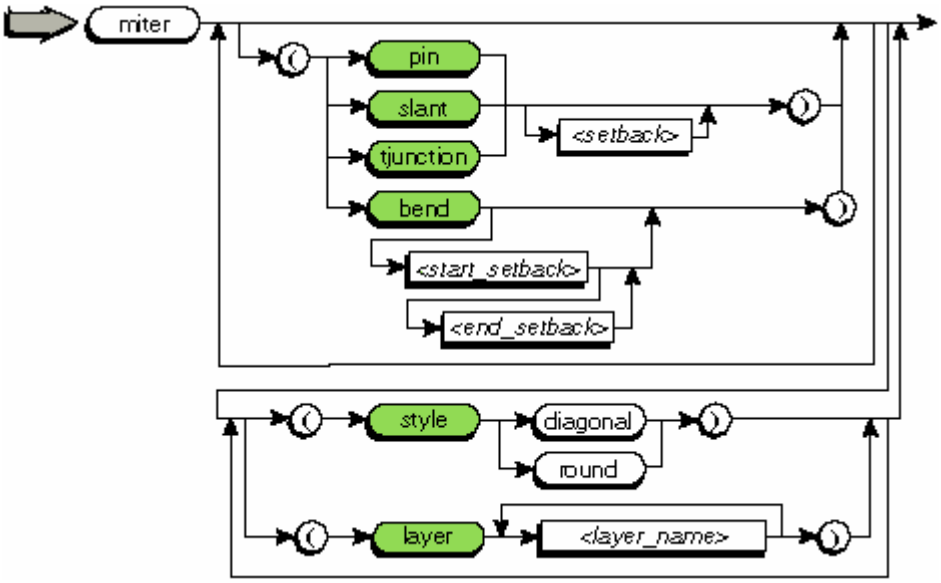
The range of **limit** values for <*positive_integer*> is 0 through 255. You can set limit values, perform some routing passes, and return to the default system values by executing a **limit** command with a value of -1. If you don't supply **limit** values, the autorouter uses default values.

Command examples

limit via 3
limit way 300
limit way -1

miter

The **miter** command changes 90 degree wire corners to 135 degrees.



pin *<setback>*

Specifies a cut or an arc at a pin (includes SMD) if the pin-to-turn distance is equal to or greater than <*setback*>. The pin setback distance is measured from the center of the pin to the turn.

slant *<setback>*

Replaces a wrong-way segment with a 135 degree segment, or with an arc, when the wrong-way length is equal to or greater than <*setback*>.

tjunction *<setback>*

Specifies changes to 135 degrees at wire tjunctions, where <*setback*> is the distance from the tjunction to the start of the cut. The default value for **miter tjunction** <*setback*> is 0.5 inch.

bend *<start_setback> <final_setback>*

Specifies a cut or an arc at a bend. The <*start_setback*> parameter specifies the initial setback distance that is attempted. When all attempts fail during this initial iteration, the value of <*start_setback*> is divided by two and the new value is attempted in all remaining 90 degree bends. This process continues. After all attempts fail, the previous setback value is divided by two, and that new value is used. When the divide-by-two operation results in a value less than <*final_setback*>, the <*final_setback*> value is applied until all miter attempts fail and the **miter bend** operation terminates. When only <*start_setback*> is supplied, <*final_setback*> defaults to the minimum wire width.

style

The **miter** styles are

- **diagonal**, which changes 90 degree corners to 135 degrees.
- **round**, which replaces 90 degree corners with an arc geometry that is fitted to the corners. When the style option is round, the pin setback value is used for all corners. If pin setback is not specified, the **miter round setback** defaults to 1 unit_line, where:

unit_line = wire width + wire_wire clearance

The unit_line value is computed by layer.

layer *<layer_name>*

Applies the miter operation to only the specified layers. If you enter multiple layer names, separate them with a blank space.

This command is similar to the **recorner** command but with enhanced functionality. When the style option is **diagonal**, corners with 90 degree bends are changed to 135 degrees. When the **miter** command is used without options, the operation applies to all miter types and defaults to style diagonal. For example, the **miter** command is the same as the following:

miter (pin) (slant) (bend) (tjunction) (style diagonal)

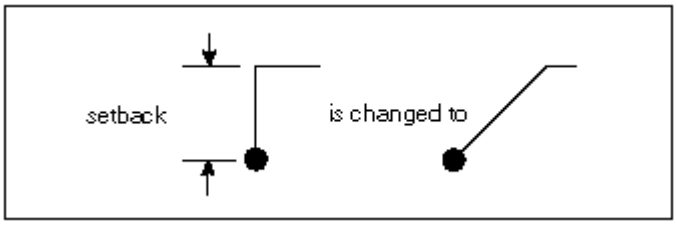
When a setback parameter is not supplied with a miter type, <*setback*> uses default values. The default pin and slant <*setback*> value is 1 inch. The default bend <*start_setback*> value is .5 inch. The default <*final_setback*> value is the minimum wire width.

During the **miter** operation, the autorouter examines all 90 degree corners and attempts to replace them with 135 degree corners, or with an arc, by using either the specified or default setback values. If at least one **miter** attempt with a given setback value is successful during a pass, the autorouter iterates with that setback value and retries all remaining 90 degree corners.

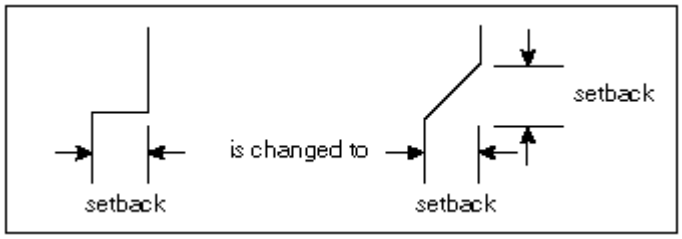
If you must apply engineering changes or route the design again, use the **unmiter** command to remove the 135 degree corners. The autorouter is more efficient when rerouting orthogonal wires.

The **miter** types are defined as follows:

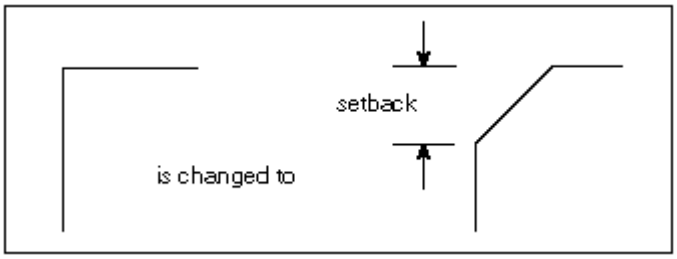
- **pin** <*setback*> specifies a cut or an arc at a pin (includes SMD) if the pin-to-turn distance is equal to or greater than <*setback*>. The pin setback distance is measured from the center of the pin to the turn.



- **slant** <*setback*> replaces a wrong-way segment with a 135 degree segment, or with an arc, when the wrong-way length is equal to or greater than <*setback*>.



- **bend** <*start_setback*> <*final_setback*> specifies a cut or an arc at a bend. The <*start_setback*> parameter specifies the initial setback distance that is attempted. When all attempts fail during this initial iteration, the value of <*start_setback*> is divided by two and the new value is attempted in all remaining 90 degree bends. This process continues. After all attempts fail, the previous setback value is divided by two, and that new value is used. When the divide-by-two operation results in a value less than <*final_setback*>, the <*final_setback*> value is applied until all miter attempts fail and the **miter bend** operation terminates. When only <*start_setback*> is supplied, <*final_setback*> defaults to the minimum wire width.



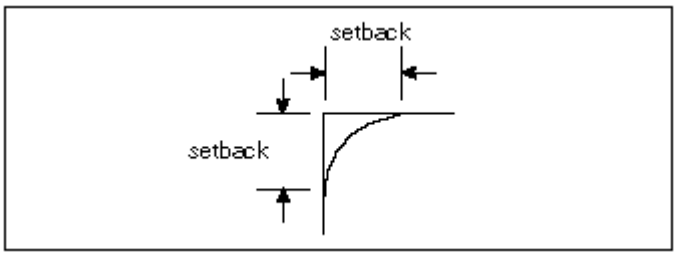
- **tjunction** <*setback*> specifies changes to 135 degrees at wire tjunctions, where <*setback*> is the distance from the tjunction to the start of the cut. The default value for **miter tjunction** <*setback*> is 0.5 inch.
- **layer** <*layer_name*> applies the miter operation to only the specified layers. If you enter multiple layer names, separate them with a blank space.

The **miter** styles are

- **diagonal**, which changes 90 degree corners to 135 degrees
- **round**, which replaces 90 degree corners with an arc geometry that is fitted to the corners. When the style option is round, the pin setback value is used for all corners. If pin <*setback*> is not specified, the **miter round setback** defaults to 1 unit_line, where:

    unit_line = wire width + wire_wire clearance

The unit_line value is computed by layer.



Note

If you apply **miter** to a differential pair that is routed on-grid, the wires may be moved to position the center point between the pair on-grid.

See also

unmiter command, which removes the 135 degree corners created by **miter**.

Command examples

miter
miter (pin 50) (slant 100) (bend 1000 50) (style diagonal)

mode

The **mode** command sets the left mouse button mode.

<interactive_routing_mode>

The keyword or keywords for the interactive [LB] mode you want to set. You can set [LB] to route, edit, move, copy, or delete wires or wiring polygons, change via attributes or wire widths, draw area outlines, select and unselect design objects, or perform other interactive operations.

<interactive_placement_mode>

The keyword or keywords for the interactive [LB] mode you want to set. You can set [LB] to place or relocate components, edit generated devices, draw area outlines, select and unselect design objects, or perform other interactive operations.

This command sets the interactive [LB] mode. You can set modes for interactive routing and for interactive placement and device editing. The current [LB] mode determines what action results when you click or drag the mouse in the work area.

To set a mode, you use the keyword(s) that identify the mode.

See draw modes for information about using [LB] to draw fences, keepout areas, regions, or rulers.

Note

The Mode Status Area along the bottom of the SPECCTRA window indicates the current interactive mode.

Command examples

mode measure
mode slide
mode copying
mode critic wire
mode change_conn
mode change_polygon
mode change_via
mode change_wire
mode cut
mode delete wire
mode merge poly_wire
mode merge keepout
mode select guide
mode edit fence

Interactive routing modes

Click on a mode option to see its description. The following list shows the keywords you use to set an interactive routing mode in the **mode** command.

change connectivity
change_polygon
change_via
change_wire
check_area
copying
copy polygon
critic wire
cut
cut_polygon

<*delete modes*>

delete keepout
delete net
delete poly_wire
delete segment
delete wire

<*edit modes*>

edit
edit fence
edit keepout
edit polygon
edit region
edit ruler

<edit topology modes>
    pick net
    pin attribute
    add virtual pin
    delete virtual pin
    move virtual pin
    reorder guides
    fix/unfix pins
    set fromto rules

highlight
measure
merge poly_wire
merge keepout
repair net
rotate_via (available only with RouteMVIA license)

<*select modes*>
    select comp
    select guide
    select net
    select pin
    select poly_wire
    select wire

slide

See draw modes in interactive routing for information about using [LB] to draw fences, keepout areas, regions, or rulers.

change_conn

Sets [LB] to change the net assignment of floating wires or wiring polygons to an existing net chosen from the Change Connectivity Setup net list dialog box.

change_polygon

Sets the [LB] to change the layer or net assignments for wiring polygons.

change_via

Sets the [LB] to do any or all of the following:

- replace a via with another type of via
- change testpoint attributes of a via.
- change fanout attributes of a via
- change the number of rows or columns in a via array. (Available only with the RouteMVIA license)

change_wire

Sets the [LB] to change the width of individual segments of routed wires.

check_area

Sets the [LB] to find and mark routing and placement violations within a rectangular bounding box. You define the bounding box by dragging with the [LB].

copying

Sets the [LB] to copy wires and vias.

You can copy an existing wire to an unroute with a similar length and path. You can also copy escape wires and vias, with their escape attributes, from a component to another component with the same image.

Note

You cannot copy a wire that belongs to a power net.

copy polygon

Sets the [LB] to copy individual objects or objects within a rectangular area. You can copy wiring polygons and keepout areas.

critic wire

Sets the [LB] to remove extra bend points in a single wire or in several wires if you draw a bounding box

cut

Sets the [LB] to divide a single wire segment into two segments. A single mouse click divides a wire segment at the cursor location. Two mouse clicks in different positions divide all the wire segments that cross a line drawn between the two locations.

cut_polygon

Sets the [LB] to cut a rectangular area out of an existing polygon. Cuts are made as follows:

- Where two or more polygons overlap, only the top polygon is cut.
- Only orthogonal and 45-degree polygons are cut.
- Polygons cannot be divided into pieces using this command.
- Connectivity is recalculated when a wiring polygon is cut.
- If a top level keepout defined in the design file is cut, the changes are saved when you write a session file.

delete

Sets [LB] to delete segments, wires, nets, or keepout areas. The choices are

delete segment, which sets the [LB] to remove a single wire segment.
delete wire, which sets the [LB] to delete all segments between two terminal points. A terminal point is a pin, via, or tjunction.

delete keepout, which sets the [LB] to remove keepout areas. You cannot remove keepouts defined in an image.
delete poly_wire, which sets the [LB] to remove wiring polygons.
delete net, which sets the [LB] to delete all wires and vias on a net. The net is not deleted.

edit

Sets [LB] to route or edit wires and wiring polygons, or to draw area objects such as keepouts or regions. The choices are

edit, which lets you route new wires or edit existing wires.

edit polygon, which lets you draw a rectangular wiring polygon.

edit fence, which lets you draw a route keepin area.

edit keepout, which lets you draw areas where you want to prohibit routing or placement. The type of keepout you draw determines what objects are prohibited.

edit region, which lets you draw the area for which you want to define certain routing rules.

edit ruler, which lets you draw a calibrated ruler anywhere within the work area.

edit topology

Sets [LB] to one of the topology editing modes. The choices are

pick_net, which lets you pick (select) a net for topology editing.

pin_attrib, which lets you assign the source, load, terminator, expose, or no expose attributes to the pins of the net you are topology editing.

add_virtual_pin, which lets you add virtual pins to the net you are topology editing.

remove_virtual_pin, which lets you delete virtual pins from the net you are topology editing.

move virtual_pin, which lets you move the virtual pins of the net you are topology editing.

reorder and **reorder_by_comp**, which let you change the order or connectivity of the net you are topology editing. You can specify starburst, daisy, mid-driven daisy, or balanced daisy net ordering.

fix_pin, which lets you disallow/allow routing to pins of the net you are topology editing.

set_rules, which lets you set fromto rules for individual fromtos of the net you are topology editing. You can set clearance, wiring, timing, shielding, crosstalk, and noise rules.

forget_fromto, which lets you remove fromto rules for individual fromtos of the net you are topology editing.

highlight

Sets the [LB] to highlight nets interactively.

measure

Sets the [LB] mode to Measure mode. You can use this mode to measure the distance between two points or extract information about routing objects and design rule violations at a specific coordinate. SPECCTRA displays measurement information in the output window, message area, and coordinate area. Object and rule information is displayed in the output window and message area.

merge keepout

Sets the [LB] to the merge keepout mode. You can merge overlapping keepout polygons that are the same type, occupy the same layers, and have the same rules within an area by sweeping the pointer across the area.

merge poly_wire

Sets the [LB] to the merge poly_wire mode. You can merge overlapping wiring polygons that belong to the same net and occupy the same layers within an area by sweeping the pointer across the area.

repair net

Sets [LB] to delete wire segments that violate fromto order rules on a net. A fromto is a user-specified pin-to-pin connection.

rotate_via

Sets the [LB] to rotate a via in ninety-degree increments.

select

Sets [LB] to select objects for routing operations, or unselect objects that are already selected. The object_type must be one of the following:

component, which sets [LB] to select or unselect components.
net, which sets [LB] to select or unselect nets.
wire, which sets [LB] to select or unselect wires.
guide, which sets [LB] to select or unselect unroutes.
pin, which sets [LB] to select or unselect component pins.
poly_wire, which sets [LB] to select or unselect wiring polygons.

slide

Sets the [LB] to move individual objects or objects within rectangular areas. You can move wire segments, vias, wire corners, polygons (wiring polygons and keepout areas), and polygon edges.

Draw modes

Use the **mode edit** command to set [LB] to a drawing mode. You can

- Draw fences in Draw Fence mode.
- Draw keepout areas in Draw Keepout mode.

- Draw regions in Draw Region mode.
- Draw rulers in Edit Ruler mode.

Drawing a fence

A fence is an autorouting keepin area, drawn as an enclosed shape consisting of corners (vertexes) connected by lines. To draw a fence in Draw Fence mode, click the location for each corner. SPECCTRA draws the lines between the corners. If the location of the last corner is not the same as the location of the first corner, SPECCTRA closes the outline for you.

You can draw a fence on a single layer or on all signal layers. Connections are routed within the fence depending on the fence setting for the design, which will be either hard (the default) or soft. See the set soft fence command for more information.

Use the fence command instead of Draw Fence mode if you want to specify the exact X and Y coordinates for each corner of the outline.

If you want to change the shape or location of an existing fence, you must delete and redefine it. Use the delete fence command to remove a fence.

Note: Hard and soft fence types cannot coexist. Either all fences in a design are hard or all are soft.

Drawing a keepout area

A keepout area is an enclosed shape consisting of corners (vertexes) connected by lines. To draw a keepout area in Draw Keepout mode, click the location for each corner. SPECCTRA draws the lines between the corners. If the location of the last corner is not the same as the location of the first corner, SPECCTRA closes the outline for you.

You can draw a keepout area on a single layer or on all signal layers. Prohibited objects cannot touch or cross a keepout outline. Use the define keepout command instead of Draw Keepout mode if you want to specify the exact X and Y coordinates for each corner of the outline.

If you want to change the shape or location of an existing keepout area, you can use the Edit Polygon mode or you can disband and redefine it. Use the forget command to disband keepout areas. You can disband any keepout area defined in SPECCTRA, but you cannot delete keepout areas defined in the image section of the design file.

Drawing a region

A region is a routing rule area drawn as an enclosed shape consisting of corners (vertexes) connected by lines. To draw a region in Draw Region mode, click the location for each corner. SPECCTRA draws the lines between the corners. If the location of the last corner is not the same as the location of the first corner, SPECCTRA closes the outline for you.

You can draw a region on a single layer or on all signal layers. Use the define region command instead of Draw Region mode if you want to specify the exact X and Y coordinates for each corner of the outline.

By default, rules assigned to a region apply to all nets in the region. Optionally, you

can define a region in which rules apply only to a specific net, to a specific class of nets, or between two classes. Use the rule command to assign rules to a region.

If you want to change the shape or location of an existing region, you must disband and redefine it. Use the forget command to disband regions.

Drawing a ruler

A ruler is a graduated scale drawn between two points in the work area. Every fifth tick mark is labeled with a distance value. To draw a ruler in Edit Ruler mode, click once at the desired start point and once at the desired end point.

You can draw horizontal, vertical, or 45 degree diagonal rulers. To draw 45 degree rulers, the Snap Angle option must be set to either 45 Degrees or All in the Interactive Routing Setup dialog box. The default Snap Angle is 45 Degrees.

net_property

The net_property command assigns properties to nets.

<property_name>

A keyword that identifies a standard property or a user property. Each property you assign must consist of a keyword (<*property_name*>) and a value (<*property_value*>). The value might be another keyword, a number, or a character string depending on what the property requires.

This command lets you assign both standard properties and user properties to one or more signal or power nets. You can either select the nets before using this command or specify the name (<*net_id*>) of each net. If you do not specify net names, SPECCTRA assigns the properties to all selected nets.

A property consists of the keyword (<*property_name*>) that identifies the property, and a value (<*property_value*>). Property values can be numbers, keywords, or character strings depending on the property.

There currently are no standard properties available for nets.

Properties can be assigned in SPECCTRA or in the design file, but a property assigned to a net in the design file cannot be changed or removed in SPECCTRA.

You can use the report command to generate a property report that contains the current values of properties assigned to all nets in the design.

Command examples

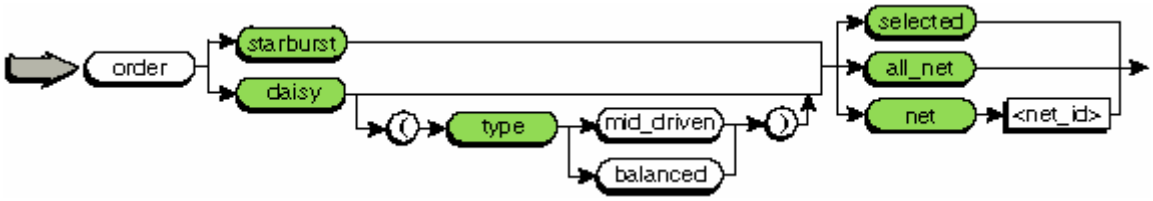
net_property sig1 (uprop_1 0.02)

net_property sig2 sig3 (uprop_2 xyz)

net_property (uprop_3 1.2)

order

The **order** command controls whether nets are routed in daisy-chain or starburst fashion.



starburst

Permits multiple entries and exits on pins.

daisy

Permits only a single entry and a single exit in the net on each pin and does not allow tjunctions. This is called a simple daisy chain. You can choose mid-driven or balanced daisy chain routing by using the **type** option.

type

Controls how a net is ordered for daisy chain routing. The choices are:

mid_driven, where a terminator is placed at each end of the net, and the loads are added back to a source. If there is more than one source, the sources are chained together first before the rest of the net is processed.

balanced, where fromtos are daisy-chained and loads are equally distributed between source and terminator pins. If more than one source pin is defined, the terminator and load branches are chained back to the closest source pin and the remaining source pins are ordered as simple daisy chain.

selected

Only the nets that are selected are ordered.

all_net

Any nets that are not fixed are ordered.

net

Orders the net named in <*net_id*>.

When SPECCTRA reads a design file, it breaks up multiple pin nets into two terminal connections. The manner in which connections are broken up depends on whether you have any daisy-chain order controls in your design file. If net order controls are not included in the design file, SPECCTRA orders all nets in starburst fashion.

The **order** command changes the original net ordering. Before you execute this

command, you must decide whether you want all nets in the design to be reordered or whether you want a different ordering for a few critical nets. You can select the nets to be ordered by using the mouse in select mode or by direct command entry.

When you execute the **order** command, the specified nets are reordered with the order **type** that you specify. The **order** command applies only to unrouted nets. Nets that are already routed cannot be ordered.

The best routing results are obtained when nets are ordered as **starburst**.

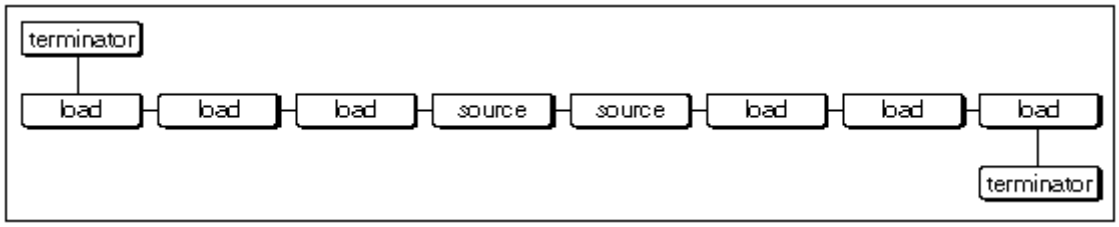
If nets in a design file include source, load, or terminator pins, but don't include a reorder control, you must execute an **order daisy** command to route them in a daisy-chain fashion from the source to load to terminator.

After you execute an assign_pin command, you must execute an **order daisy** command to reorder the pins you assigned source, load, and terminator properties.

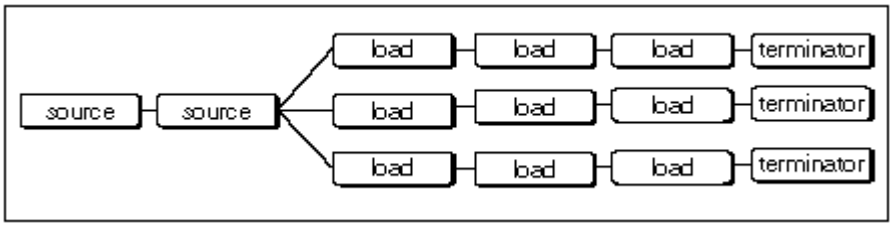
When you have fromtos ordered in a design file, you must use the forget net command before you can use the **order** command to reorder those fromtos. Remember, **forget net** disbands all net rules.

Command examples

order daisy (type mid_driven)

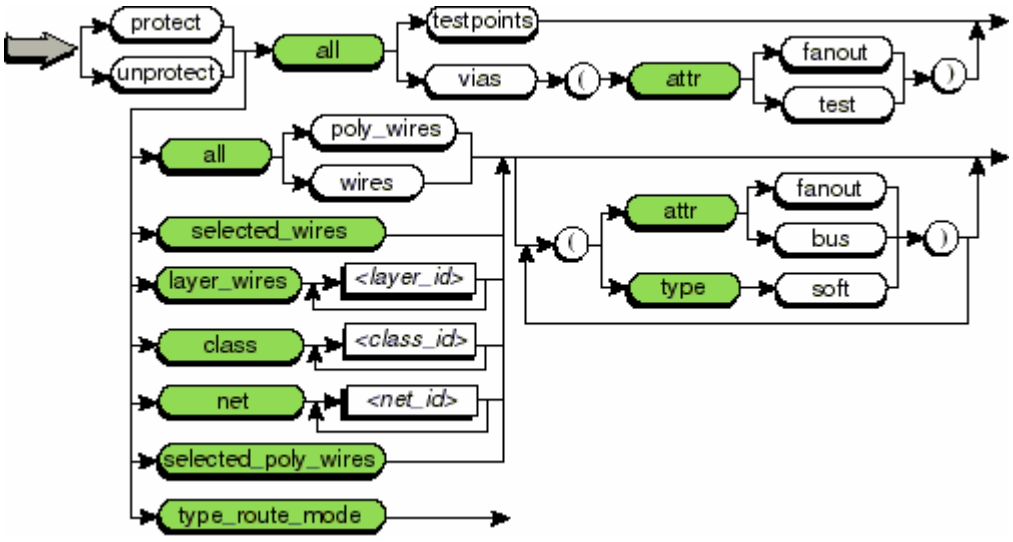


order daisy (type balanced)



protect/unprotect

The **protect** command prevents the autorouter from ripping-up and rerouting existing wires, and vias. The **unprotect** command reverses **protect**.



all

Protects/unprotects the following choices:

testpoints, which includes all test points

vias, which includes all vias

Use the **testpoints** option to specify all testpoints and vias inserted or marked by the testpoint rule. Use the **vias** option to specify all types of vias.

all poly_wires

Use the **all poly_wires** option to protect/unprotect wiring polygons. This command has no affect on wires. See **all wires** instead.

all wires

Protects/unprotects wiring. Use the **all wires** option to protect/unprotect routed wires. This command has no affect on poly_wires. See **all poly_wires** instead.

selected_wires

Protects/unprotects only the wiring that is currently selected. No other selected routing objects are protected or unprotected.

layer_wires

Protects/unprotects all routed wires on layer <*layer_id*>. Multiple layer names can be included.

class

Protects/unprotects all routed wiring of nets included in class <*class_id*>. Multiple class names can be included.

net

Protects/unprotects all routed wiring for the net <*net_id*>. Multiple net names can be included.

selected_poly_wires

Protects/unprotects only the wiring polygons that are currently selected. No other selected routing objects are protected or unprotected.

attr

Protects/unprotects only those vias with the named attribute. If multiple attributes are assigned to a via, you can protect/unprotect that object by using any one of the attributes.

Use the **fanout** option to protect/unprotect only the vias created with the fanout command.

Use the **test** option to protect/unprotect all vias added by the testpoint command.

attr

Protects/unprotects only those wires with the named attribute. If multiple attributes are assigned to a wire, you can protect/unprotect that object by using any one of the attributes.

Use the **fanout** option to protect/unprotect only the wires routed with the fanout command.

> **Note**: This option does not protect wires routed interactively from fanout vias created with the **fanout** command or translated from the host layout system.

Use the **bus** option to protect/unprotect only the wires routed with the bus command.

type soft

Protects/unprotects all wires and vias that the autorouter can push and shove when space is needed for other routing.

type_route_mode

Controls whether wires and vias defined as **type route** in the design file are unprotected when you use subsequent **unprotect** commands to unprotect wires or vias.

- Using **unprotect type_route_mode** means that wires defined as **type route** can be unprotected by subsequent commands.
- Using **protect type_route_mode** means that wires defined as **type route** cannot be unprotected by subsequent commands.

The default is that wires defined as **type route** cannot be unprotected by subsequent commands.

Use the **protect** command to protect preroutes and other design objects that you

want to preserve. You can also use the command when you want to preserve fanout or bus routing or to preserve the routing after you read wires from an external file. Use the **unprotect** command to remove the protect status from objects.

Note

Protect and **unprotect** apply to routed wires. See the fix and unfix commands to control routing of nets.

Unprotect does not affect wires marked as (type fix) in the design file.

Command examples

protect all wires
protect all wires (attr fanout)
protect all wires (attr bus)
unprotect all wires (attr bus)
unprotect selected_wires

protect net CLK1
unprotect net CLK1

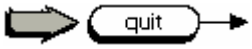
protect layer_wires s2 s3
unprotect layer_wires L2 L3

protect all vias (attr fanout)
unprotect all vias

protect all testpoints
unprotect all testpoints

quit

The **quit** command exits SPECCTRA.



The **quit** command terminates SPECCTRA operation. If you have unsaved changes in the design, SPECCTRA prompts you to save changes in a session file and provides the option of deleting the current did file. If you have no unsaved changes, you are offered the option of deleting the current did file.

The **quit** command can be entered from the Command entry area or from a do file.

You can also quit by clicking Quit on the File menu.

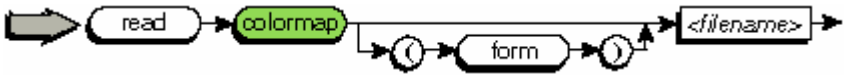
If you use the -quit switch when you start SPECCTRA, operation immediately terminates after the last command executes in the start-up do file.

Command example

quit

read colormap

The **read colormap** command loads a color map file.



colormap

Loads a previously saved colormap from the named file. The **form** option loads the colormap into the current palette instead of into the session and displays the Load Colormap dialog box for interactive adjustments of color and pattern selections.

When you use this command, SPECCTRA reads the named color map file. The color map file contains data that defines the display colors and patterns for design objects and graphical features in the work area.

Note

For general information about specifying filenames, see File Naming Conventions.

See also

write colormap

Command Example

read colormap color1.std

read keepout

The **read keepout** loads keepouts from a session file.

keepout

Loads top-level keepouts from a session file that contains data from a previous routing session.

When you use this command, SPECCTRA loads top-level keepouts that are in the session file. Only keepouts that you add, modify, or delete are saved in the session file. Top-level keepouts are keepouts defined in the structure section of the design file or session file.

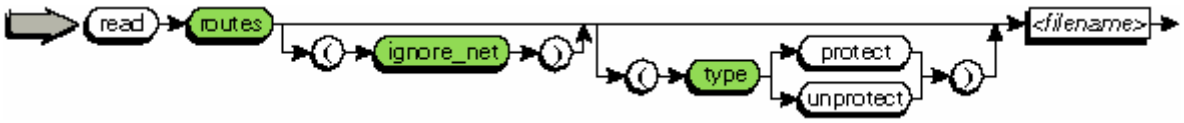
Command example

read keepout design.ses

For general information about specifying filenames, see File Naming Conventions.

read routes

The **read routes** command loads a routes file.



routes

Loads a routes file that contains data for all routed wires and vias, plus additional information for translating the route data back to the host layout system.

ignore_net

Disables the use of net names recorded in the Net_out section of the routes file, and enables SPECCTRA to determine net names based on the pins, wires, and vias on the design.

type

Limits the wires read from the routes file to:

protect, which reads only protected wires
unprotect, which reads only unprotected wires

This command reads files that are created with the **write routes** command. When you read a routes file, any existing wires are replaced by wires in the routes file. If you don't want to merge the wires in the routes file with existing wires, use delete all wires before you execute **read routes**.

If you use write session and restart SPECCTRA with a session file, you don't need to read the routes file in a separate operation.

When you change a netlist in your layout system, you can apply engineering change orders (ECOs) in the autorouter by loading your design file with the wires or routes file. You can use the **read routes** command with the **ignore_net** option to load the routes file for ECO processing. This disables the use of net names recorded in the Net_out section of the routes file, and enables SPECCTRA to determine net names based on the pins, wires, and vias on the design.

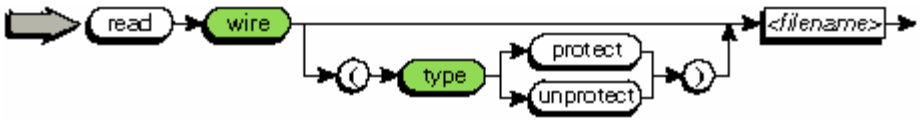
Command examples

read routes rev_c.rte
read routes (ignore_net) rev_d.rte

For general information about specifying filenames, see File Naming Conventions.

read wire

The **read wire** command loads a wires file.



wire

Loads a wires file that contains data for all routed wires and vias.

type

Limits the wires read from the wires file to:

protect, which reads only protected wires
unprotect, which reads only unprotected wires

You can read wires from an external file and add the wires file data to existing wiring. Any existing wires that are redundant with wires in the wires file are replaced. If you don't want to merge existing wires, use delete all wires before you execute **read wire**.

Use a **delete all wires** and **read wire** command sequence to view the routing results from different autorouting sessions.

Note

The preferred method of reading wire data during autorouting is to specify the wires file with the Startup dialog box.

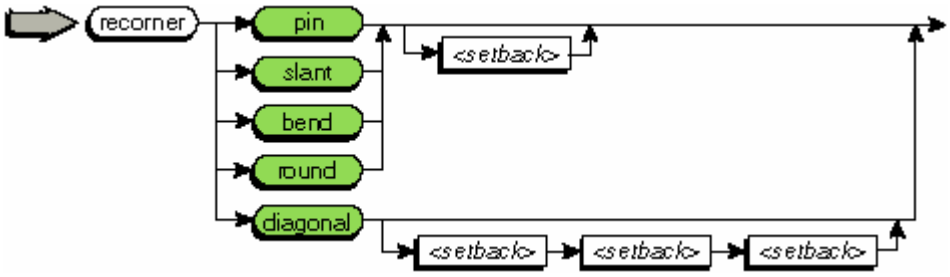
Command example

read wire rev_a.w

For general information about specifying filenames, see File Naming Conventions.

recorner

The **recorner** command changes 90 degree wire corners to 135 degrees. See the **miter** command for improved function.



pin

Changes wire corners at pin and via exits to 135 degrees if the corner occurs at or above the *<setback>* distance. Rectangular pads are excluded from this operation.

slant

Replaces two 90 degree corners with two 135 degree corners.

bend

Changes a 90 degree corner to a 135 degree corner.

round

Replaces 90 degree corners with arcs.

diagonal

Performs all pin, slant, and bend operations. The three *<setback>* values are for pin, slant, and bend, respectively.

<setback>

The *<setback>* value is a positive dimension you provide. If you do not provide a *<setback>* value, the following default values are used:

pin, slant	1 inch
bend	0.5 inch
round	Sum of wire width plus wire_wire clearance

The **recorner** command changes corners from 90 to 135 degrees to improve manufacturability. The round option, which replaces square corners with arcs, is available only with a fast-circuit license. The **pin**, **slant**, and **bend** options control which corner locations are changed. If <*setback*> is not supplied, default <*setback*> values are used. The <*setback*> value must be a positive dimension. Each corner is checked before chamfering to avoid creating new conflicts.

The **recorner diagonal** command performs pin, slant, and bend operations simultaneously. If you enter the **recorner diagonal** command without setback values, the autorouter uses default setback values.

If you apply engineering changes or reroute the design, use the unmiter command to remove the 135 degree corners. The autorouter is more efficient when it is rerouting orthogonal wires.

For illustrations of the **recorner** options, see the miter command.

Setback is rounded up to the nearest wire grid dimension unless a wire grid is not specified (gridless). If the setback for a round corner is too large for the arc to be completed, the setback distance is reduced until the arc fits.

Usually, the **recorner** command is executed as the last step in the autorouting process, just before routing is returned to the host system.

Command examples

```
recorner bend 0.250
recorner diagonal 0.5 0.5 0.5
```

undo/redo

The **undo** command reverses interactive routing, editing, and placement operations. The **redo** command reapplies interactive operations that were reversed by **undo**.

You can reverse a single interactive operation by entering the **undo** command or by using `[F3]` or `[Undo]`.on your keyboard. You can also reverse a series of operations by entering a series of **undo** commands.

You can immediately **redo** an operation that was reversed by the **undo** command. You can also redo a series of undo operations by entering multiple redo commands or by using shortcut keys. The shortcut keys to redo an operation are `[Shift] [F3]` or `[Shift] [Undo].`

The interactive routing and editing operations that can be reversed with **undo** and reapplied with **redo** are

Add/Edit Polygon
Change Connectivity
Change Polygon
Change Via
Change Wire
Copy Polygon
Copy Route
Critic Route
Cut Segment
Cutout Polygon
Delete (all modes except Repair Net)
Edit Route
Merge Wiring Polygon
Move
Select/Unselect (except pins)

If there is no command operation in memory to undo, an information dialog box appears with the message

```
Nothing (more) to undo.
```

Note

Repair Net operations, Edit Topology operations, and Select/Unselect gate, subgate, pin, and terminator operations cannot be reversed by **undo** or reapplied by **redo**.

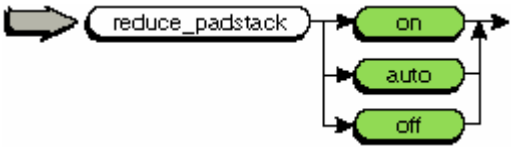
Command example

unplace all
undo

redo

reduce_padstack

The **reduce_padstack** command controls whether smaller layer shapes are substituted for through-pins.



on

The **reduce_padstack on** command directs the autorouter to immediately substitute the alternate shapes regardless of the conflict reduction status. Once the alternate shapes are substituted, you can't restore the larger shapes unless you restart SPECCTRA.

auto

The **reduce_padstack auto** command can be used if you expect the autorouter to have difficulty converging to a 100% solution. The autorouter monitors progress and substitutes the smaller padstacks if the conflict reduction rate is too low.

off

The **reduce_padstack off** command turns off the **reduce_padstack auto** function. This command is effective only if the autorouter has not already substituted alternate shapes. Once alternate shapes are substituted, **reduce_padstack** cannot be turned off. The **reduce_padstack** command defaults to **off** if not specified.

The **reduce_padstack** command frees critical routing space on dense PCBs. When you execute **reduce_padstack**, the autorouter substitutes alternate, smaller padstack shapes on certain layers. The substitution applies only to through-pins, and the alternate padstack shapes must be included in the design file. The smaller shapes are substituted by layer only where there are no connections to the default shapes on a layer. The smaller shapes free routing space that is critical to completing a difficult PCB.

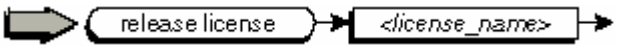
For additional information, see <*reduced_shape_descriptor*> in the *Design Language Reference*.

Command examples

reduce_padstack on
reduce_padstack auto
reduce_padstack off

release license

The **release license** command checks a license back in to the pool of available licenses for all SPECCTRA users.



Use this command to release a feature license from a SPECCTRA session without ending the session. A license released in this way becomes available for other SPECCTRA users.

The following list shows the valid license names for SPECCTRA features. License names for your version of SPECCTRA will be either standard format or RPP format. Use the appropriate style for <*license_name*>, or use the abbreviation.

Tip

Use the license_usage command to see a list of licenses currently checked out to your SPECCTRA session.

Note

If you use “all” for <*license_name*>, all licenses except ViewBase are checked in. ViewBase is the minimum license required to keep the session running.

Command example

release license RouteADV

repaint

The **repaint** command refreshes the work area portion of the SPECCTRA window.

When you enter the **repaint** command, all visible layers are redrawn in the order they appear in the layer panel, from bottom to top.

If you are routing interactively and have set the active and alternate layers, those layers are drawn on top.

Tip

You can press the [ESC] key (escape), when the mouse pointer is in the work area, to halt screen repainting.

Notes

You can use the **repaint** option in the set command to disable or enable all repaint operation, or to permit repaints only when you explicitly perform a viewing operation (such as zoom, pan, or repaint). All repaints are enabled by default.

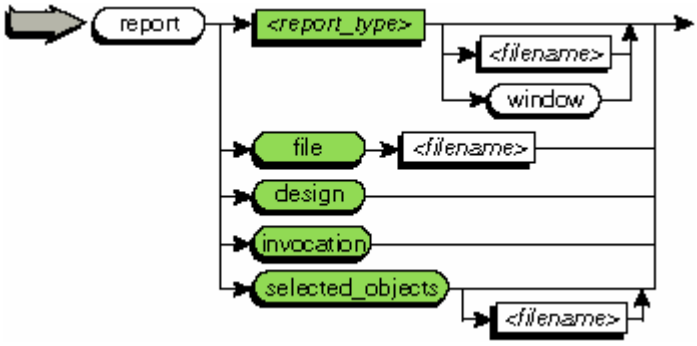
You can also use the **dofile_auto_repaint** option in the **set** command to control whether SPECCTRA repaints the work area after operations performed by commands in a do file.

Command example

repaint

report

The **report** command generates a routing or placement report.



<report_type>

You use a report keyword to generate a specific report with the report command. You can generate the following reports:

class
class_class
component <*component_id*>
corners
crosstalk
ecl
group
group_set
keepouts
layer <*layer_id*>
length
net bundles
net <*net_id*>
no_fanout
order_violations [(**no_stubs**)]
padstack
pairs
power_fanout_order_violations
property (<*property_objects*>)
regions
stack_via_depth
status
testpoint
unconnect
vias

file

Displays the specified file in the report window. You must specify the name of an existing text file.

design

Displays the current design file in the report window. You cannot save the design to a new file, and you cannot run **report design** from a do file.

See the *Design Language Reference* manual for a description of the syntax used in the design file.

invocation

Lists all error and warning messages generated when starting SPECCTRA.

selected_objects

Lists information about selected objects. You must select one or more of the nets, components, images, guides, wires, and pins that you want information about.

The net information includes name, number of pins, vias, wires, and tjunctions for each net, and routing length data.

The component information includes name, rotation, layer placed on, X and Y coordinate location, part number, and type for each component.

The image information includes name and number of pins for each image, and reference designator of each component instance.

The guide information includes fromto data, Manhattan length of unrouted connections, and actual length of the routed portion of unfinished connections for each guide.

The wire information includes fromto data identified by component reference designator and pin number, or object type, and X and Y coordinate location for each wire segment, and layer on which wire segments are routed.

The pin information includes component reference designator and pin number, or object type, X and Y coordinate location, and padstack for each pin, and layers on which pins are connected.

The default report filename is selobj.rpt.

See selected objects report for detailed descriptions of the information contained in this report.

This command displays a placement or routing report in the report window, saves a report in a text file, or displays a text file in the report window.

- Use a report type to generate a report about a placement or routing object or a current design condition. See <*report_type*> for a description of the different reports you can generate.
- Use **file** to display the contents of a text file in the report window.
- Use **design** to display the design file in the report window.
- Use **selected_objects** to display a list of all currently selected placement and routing objects in the report window.
- Use **invocation** to display a list of startup errors in the report window.

When you generate a report, you can

- Use **window** to display the report in the report window.
- Use <*filename*> to save the report to a specific directory with a specific filename.

Each <*report_type*> has a default filename. If you do not include either a filename or the **window** keyword, SPECCTRA uses the default filename and saves the report in the design directory. You must supply a filename to save a report file in a different directory. See file naming conventions for related information.

Notes

The component, net, netlength, and layer reports provide information about a particular component, net, or layer. The eco report provides information about changes between a particular design and another iteration of the design. To generate one of these reports

- Specify the component reference designator (<*component_id*>)
- Specify the net name (<*net_id*>)
- Specify the layer name (<*layer_id*>)
- Specify the design name followed by the changed design name (<*old.dsn*> <*new.dsn*>)

Command examples

```
report class
report file board3.do
report design
report net sig18 sig18.rpt
report selected_objects
```

<*report_type*>

class

Lists all defined classes and the nets contained in each class.

The default report filename is classes.rpt.

To list current class rules, use the report rules command.

class_class

Lists rules assigned to all defined class-to-class pairs.

The default report filename is clscls.rpt.

You can also list current class-to-class rules, by using the report rules command.

component

Lists component type, image name, side, rotation, and location information about a component. You can either select the component or specify its reference designator (<*component_id*>). This report lists all placement rules that currently apply to the component, and includes assigned image and component properties.

This report also lists information for each component pin, including position, padstack, net name, and assigned image pin and component pin properties.

The default report filename is comp.rpt.

See component report for descriptions of the information contained in this report.

corners

Summarizes the status of all routed corners in the design, listing corners that are 90 or 135 degree angles, arcs, and other angles.

It identifies how many 90 degree corners remain after running recorner or miter commands.

The default report filename is corners.rpt.

crosstalk

Lists the parallel and tandem segment crosstalk and noise rules in effect, indicates rule violations, and lists the amount of overlap. The rule violation information includes location, and the net names, pin-to-pin connections, and signal layers involved.

When you generate this report, SPECCTRA also indicates crosstalk violations graphically by a white box between offending wire segments. The long side of the box runs the length of the rule violated.

The default report filename is xtalk.rpt.

ecl

The emitter coupled logic (ecl) report lists net order violations with pin names and the routed lengths between source and terminator pins.

The default report filename is net_ecl.tmp.

group

Lists all currently defined groups of fromtos. Data is listed by group and includes group names, net names, and the pin-to-pin connections assigned to the group.

The default report filename is group.rpt.

To list current group rules, use the report rules command.

group_set

Lists the number of defined group sets, and includes the names of the groups in each group set.

The default report filename is grpset.rpt.

To list current group set rules, use the report rules command.

keepouts

Lists all defined keepouts, and includes type, shape, layer, and coordinate information for each keepout.

The default report filename is keepouts.rpt.

See keepouts report for descriptions of the information contained in this report.

layer

Lists layer properties and their values assigned to a layer. You must specify the layer name (<*layer_id*>).

The default report filename is layer.rpt.

length

Lists all nets that have length or delay rules, the current values of these rules, the actual routed length or timing delay of each net, the total violations, and an error message for each net or fromto violating the rules.

This report also includes length factor, effective via length, and pair average length information.

The default report filename is lengths.rpt.

See length and delay rules report for a general description of this report.

net

Lists information about a net, including name, fixed status, classes the net is assigned to, number of pins, vias, wires, tjunctions, and routing length data for the specified net. You must specify the net name (<*net_id*>). This report lists all rules that currently apply to the net, and includes assigned net properties. The net report also contains a network, connections, and routing section for each net.

The default report filename is net.rpt.

See net report for descriptions of the information contained in this report.

You can also list current net rules, by using the report rules command.

net bundles

Lists each net or fromto in defined bundles (busses) and their layer gaps. If a bundle gap is not specified, SPECCTRA uses the largest wire-to-wire clearance rule of the nets comprising the bundle, and the report states

```
No Bundle Gap Specified
```

The default report filename is bundles.rpt

no_fanout

Lists all component pins that lack an escape wire and via after the last fanout command runs. The pin information includes pin reference, X and Y location, padstack ID, and associated net name. Only pins that match the last used **pin_type** option in the **fanout** command appear in the report.

You can use this report to determine whether pins failed the fanout operation. You can further determine whether pins are blocked or cannot escape due to rule settings.

The default report filename is nofanout.rpt.

See pins without vias report for descriptions of the information contained in this report.

order_violations

Lists order violations and stub length rule violations (or just order violations if you use the **no_stubs** option) by net ID and the X,Y coordinate locations where the violations occurred.

The default report filename is order_viols.tmp.

padstack

Lists the via, pin, and SMD padstacks from the library section of the design file. See the *Design Language Reference* for descriptions of syntax for padstack properties.

The default report filename is padstack.rpt.

To list current padstack rules, use the report rules command.

pairs

Lists each net or fromto in defined differential pairs and their pair gap. If a pair gap is not specified for a differential pair, SPECCTRA uses the wire-to-wire clearance rule and the report states

```
No Pair Gap Specified
```

The default report filename is pairs.rpt.

power_fanout_violations

Lists all fanned-out pins that violate current power fanout rules and reports the total number of violations.

The default report name is pwr_fan_order_viol.rpt.

property

Lists object properties and their current values. The report lists object names, property types (system or user), property names, and property values. You must specify one or more object types (<*property_objects*>) to include in the report. The choices are

component lists the properties assigned to each component in the design.

component_pin lists the properties assigned to each component pin in the design.

image lists the properties assigned to each image in the design.

image_pin lists the properties assigned to each image pin in the design.

The default report filename is property.tmp.

regions

Lists all defined regions, and includes type (region, net region, class region, or class_class region), shape, layer ID, and X and Y coordinates for each region.

The default report filename is regions.rpt.

See regions report for descriptions of the information contained in this report.

To list current region rules, use the report rules command.

stack_via_depth

Lists violations of the stack_via_depth rule

status

Lists a summary of routing data for the design, and includes the following categories:

- Routing status
- Routing history
- Wiring statistics
- Summary statistics by layer

In addition to this report, the autorouter creates simplified routing statistics and displays them in the output window and saves them in a default file, *monitor.sts*, at the end of each routing pass.

SPECCTRA automatically updates the status file after every 100 wires are routed.

The default report filename is status.rpt.

See routing status report for explanations and examples of the information contained in this report.

testpoint

Lists test point summary information such as the number of nets that do not have test points, the number of test points on each side of the PCB (front and back), the size of the test point grid, and the current test point spacing and clearance rules. This report also lists information for each test point, such as location, type, layer, padstack name, pin or via name, and name of the net the test point is assigned to. It also contains the measurement units used in the design.

The testpoint report also includes a list of nets that have no **testpoint** rule in effect, and also nets that do have a **testpoint** rule but that SPECCTRA cannot find a test via site for. Since the testpoint feature is disabled for differential pairs, you can see a list of missing test points for differential pairs in this report.

The default report filename is tstpt.rpt.

See testpoints report for descriptions of the information contained in this report.

Note

Use the **highlight testpoint_violations** command to highlight test points that violate current testpoint rules.

unconnect

Lists all unconnected fromtos by net name. It includes the reference designator, pin number, and coordinate location for each pin in the fromto.

The default report filename is unconn.rpt.

vias

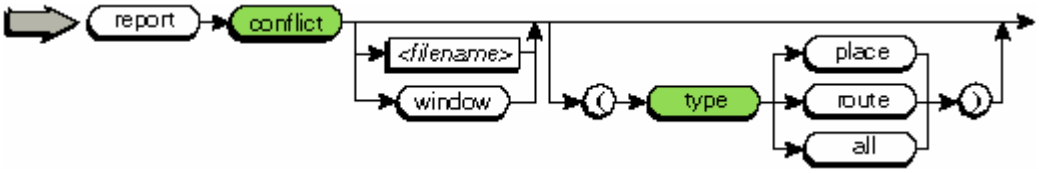
Lists all vias defined in your design file for use during automatic or interactive routing. The report includes the following information for each via:

- The layers on which the via can reside
- Whether the via is selected for routing
- The bounding box dimension (outline) for the via
- The via image shapes on each layer that define each via

The default report filename is vias.rpt.

report conflict

The **report conflict** command generates a report that contains information on current conflicts in the design.



conflict

Lists routing clearance and crossover conflicts and rule violations.

The autorouter checks all routed wires and displays a conflict shape in the graphics display area. A diamond shape represents a crossover conflict. A rectangular shape represents a clearance conflict.

You can use the **type** option to report and display conflicts and rule violations for routing, placement, or both routing and placement.

The default report filename is conflict.rpt.

type

Identifies the types of conflicts you want to include in the conflict report. The choices are

place, which lists components that violate placement rules and includes a summary of the types of violations.

route, which lists wiring conflicts.

all, which lists both placement rule violations and wiring conflicts.

This command displays a conflict report in the report window or saves the report in a text file. The default conflict report contains information on both placement and routing conflicts. To include placement conflicts or routing conflicts only, use the **type** option.

When you generate a conflict report, you can

- Use **window** to display the report in the report window.
- Use <*filename*> to save the report to a specific directory with a specific filename.

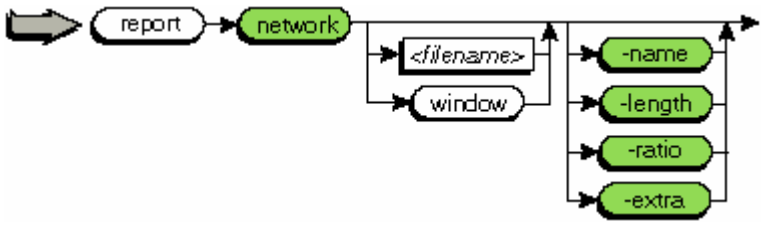
If you do not include either a filename or the **window** keyword, SPECCTRA uses the default filename, *conflict.rpt*, and saves the report in the design directory. You must supply a filename to save a report file in a different directory. See file naming conventions for related information.

Command examples

report conflict
report conflict conflct8.rpt
report conflict (type route)

report network

The **report network** command generates a report that contains the netlist.



network

Lists net names, number of pins, vias, wires, tjunctions in each net, and Manhattan versus routed lengths data for each net (including one-pin nets). You can choose the way these statistics are presented by using the **-name**, **-length**, **-ratio**, or **-extra** keywords.

The default report filename is network.rpt.

See network report for descriptions of the information contained in this report.

-name

Sorts the information about the nets alphabetically according to the net name.

-length

Sorts the information about the nets from the highest to the lowest length rule.

-ratio

Sorts the information about the nets from the highest to the lowest ratio of the actual

routed length divided by the Manhattan length.

-extra

Sorts the information about the nets from the highest to the lowest difference between the actual routed length and the Manhattan distance.

This command displays a network report in the report window or saves the report in a text file.

The default network report sorts net information by name. To sort net information by length, ratio or extra, use the sorting keywords.

When you generate a network report, you can

- Use **window** to display the report in the report window.
- Use <*filename*> to save the report to specific directory with a specific filename.

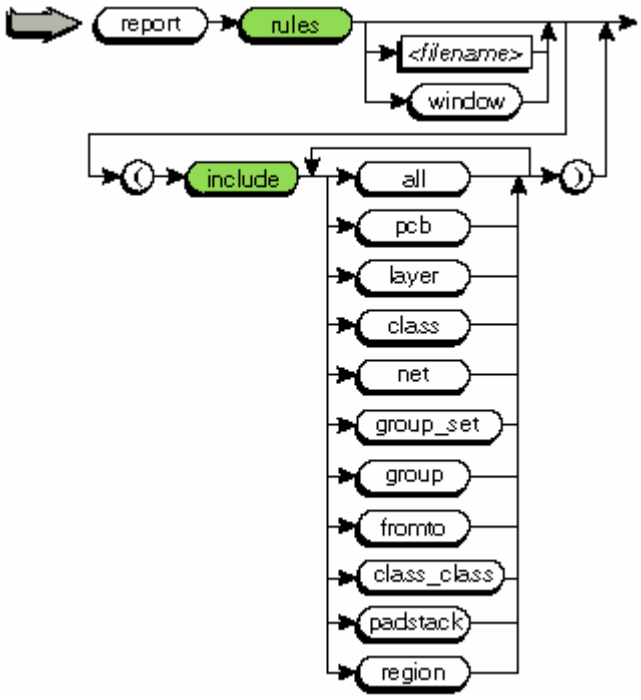
If you do not include either a filename or the **window** keyword, SPECCTRA uses the default filename, *network.rpt*, and saves the report in the design directory. You must supply a filename to save a report file in a different directory. See file naming conventions for related information.

Command examples

report network
report network -length
report network brd1.rpt
report network brd2.rpt -ratio
report network window -extra

report rules

The **report rules** command generates a report that contains the current design rules.



rules

Lists design rules currently in effect or rules that apply at specific precedence levels of the rule hierarchy that you specify by using the **include** option. Clearance rules are listed separately for each object-to-object setting. This report also contains the name of the design file, the number of signal and power layers, and the size of the via and wire grids.

The default report filename is rules.rpt.

include

Specifies which rules you want included in the rules report. You can

- List current design rules (**all**).
- List current design rules that apply at the **pcb**, **layer**, **class**, **group_set**, **net**, **group**, **fromto**, **class_class**, or **padstack**, **region** precedence levels of your design.

For more information about rule precedence, see routing rule hierarchy.

This command displays a rules report in the report window or saves the report in a text file.

The default rules report contains information on pcb and layer rules only. To include rules at other levels, use the **include** option.

When you generate a rules report, you can

- Use **window** to display the report in the report window.

- Use <*filename*> to save the report to a specific directory with a specific filename.

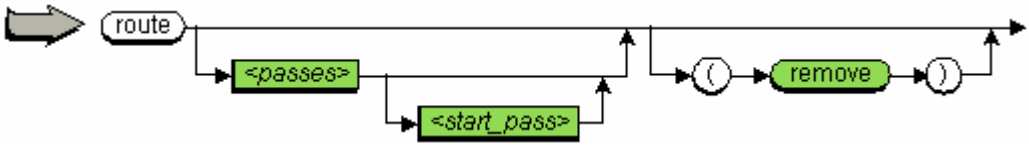
If you do not include either a filename or the **window** keyword, SPECCTRA uses the default filename, *rules.rpt*, and saves the report in the design directory. You must supply a filename to save a report file in a different directory. See file naming conventions for related information.

Command examples

report rules
report rules newrules.rpt
report rules (include class)
report rules (include net group)

route

The **route** command starts the autorouter.



<passes>

Specifies the number of wiring passes you want the autorouter to run.

Twenty-five passes is usually the suggested minimum.

<start_pass>

Sets a point in the autorouting cost table that the autorouter uses to start the series of route passes. Typical values are

1, which means the autorouter uses the costing that is used when SPECCTRA initially routes a design.

6, which means the autorouter uses the costing that is used after the initial five route passes. The cost of conflicts is relatively low at this point in the cost table.

11, which means the autorouter uses the costing that is used after the initial 10 route passes. The cost of conflicts is moderate at this point in the cost table.

16, which means the autorouter uses the costing that is used after the initial 15 route passes. The cost of conflicts is relatively high at this point in the cost table.

If you do not supply <*start_pass*>, the autorouter uses a value that is based on the completion level of the routing.

Do not use the <*start_pass*> option unless you are an experienced SPECCTRA user.

remove

Creates an unroute when the autorouter tries to reroute a wire and cannot find a new path, rather than restoring the wire to its original position.

Use this option only when the number of failures is greater than 100 and there are hundreds or thousands of conflicts after 10 or more route passes.

The **remove** option runs automatically, if the autorouter detects a poor convergence rate and failures are greater than 50. Nets with a routing priority of 200 or higher are not ripped up and removed.

You can use the **route** command at any time except in pause mode. You can use **route** without a pass number to run a single autorouting pass, or you can specify a number of autorouting passes. You use the **route** command to

- Start the initial autorouting of a PCB
- Specify the number of routing passes
- Specify a starting point (<*start_pass*>) in the autorouting cost table, which allows you to restart where you left off in a previous session
- Control whether wires involved in conflicts are removed and left as unroutes
- Constrain the autorouter to route within a certain area of your design

SPECCTRA uses the number of route passes you specify as long as conflicts remain or connections are unrouted. Once wiring is 100 percent complete with no crossing or clearance violations, unused route passes are skipped. If there are crosstalk or maximum and minimum length violations, route passes continue until these violations are also resolved.

The routing progress indicator monitors and displays the progress of the **route** command using a traffic light icon. You can click on the icon to display detailed information in a dialog box.

If you select one or more connections, the autorouter attempts only those you have selected. If no connections are selected, the autorouter attempts route or reroute all connections defined in the network except those that are fixed or protected.

How connections are routed, or how they are ripped up and rerouted, depends on the number of route passes completed in your current session and whether you include a <*start_pass*> value. During the first five route passes in an autorouting session, all connections are ripped up and rerouted if they are not fixed nets or protected wires. After the first five passes, the connections that get routed are those that are not already routed. Wires involved in conflicts, and those close to wires involved in conflicts, can be ripped up and rerouted if they are not protected.

Use **remove** to remove wires that are involved in conflicts and leave them as unroutes. Nets with a routing priority of 200 or higher are not removed by this option. Connections with high speed rules are automatically assigned a priority greater than 200. The **route** command uses **remove** automatically if the autorouter detects a poor convergence rate and failures are greater than 50.

The autorouter operates as an orthogonal router by default except in areas that include objects such as staggered pins, where the autorouter can use diagonal routing. You can change how the **route** command uses diagonal routing by using the **set** command. See set diagonal_mode for more information about controlling diagonal routing. Choose Contents and Index from the Help menu for more

information about using the **route** command.

See also

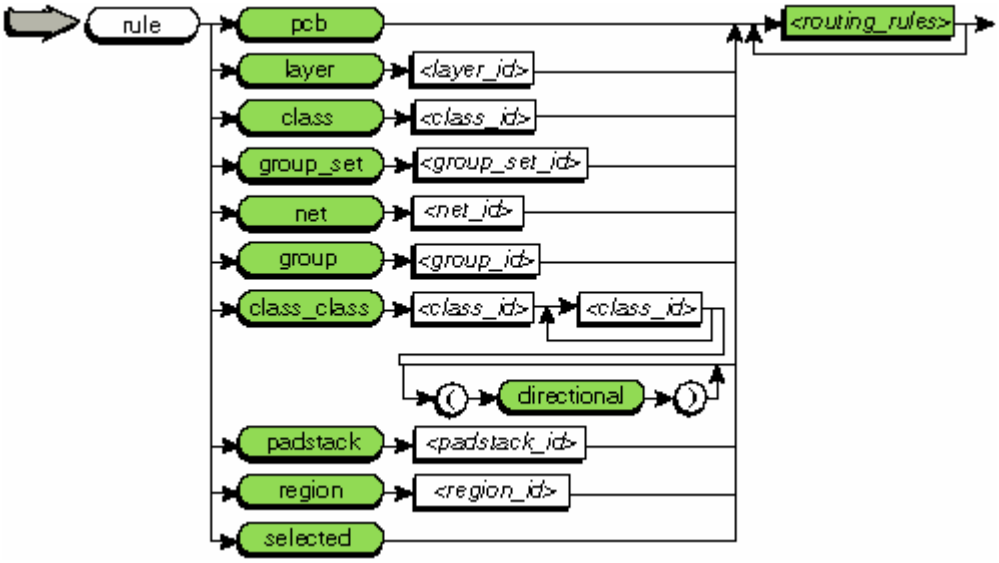
bus
fanout
smart_route

Command examples

route 25
route 50 16
route 5 (remove)

rule

The **rule** command sets routing rules at different precedence levels of the rule hierarchy.



pcb

Applies routing rules to the design.

You can apply clearance, width, wiring, timing, crosstalk, and noise rules.

layer

Applies routing rules to the specified layer. The <*layer_id*> is either the name of a signal layer or power layer defined in the design file, or one or more three possible keywords (**pcb**, **signal**, **power**).

You can apply clearance, width, wiring, time_factor, crosstalk, noise, noise_weight, and costing rules.

class

Applies routing rules to the specified class. The <*class_id*> is the name of a class defined in SPECCTRA or in the design file.

You can apply clearance, width, wiring, timing, shielding, crosstalk, and noise rules.

group_set

Applies routing rules to the specified group set. The <*group_set_id*> is the name of a group set defined in SPECCTRA or in the design file.

You can apply clearance, width, and timing rules .

net

Applies routing rules to the specified net. The <*net_id*> is the name of a net defined in the design file.

You can apply clearance, width, timing, shielding, crosstalk, and noise rules.

group

Applies routing rules to the specified group. The <*group_id*> is the name of a group defined in SPECCTRA or in the design file.

You can apply clearance, width, wiring, timing, shielding, crosstalk, and noise rules .

class_class

Applies routing rules between the specified classes. The <*class_id*> is the name of a class defined in SPECCTRA or in the design file.

You can apply clearance, crosstalk, and noise rules.

directional

The **directional** keyword determines which class is noise transmitter or noise receiver. Direction is used only for *parallel noise descriptors* and *tandem noise descriptors*. The rule applies to the pair in the order the classes are specified. Do not use **directional** when applying crosstalk rules between the wires of a single class.

padstack

Applies routing rules to the specified padstack. The <*padstack_id*> is the name of a padstack defined in the design file.

You can apply clearance rules .

region

Applies routing rules to the specified region. The <*region_id*> is the name of a region defined in SPECCTRA or in the design file.

You can apply clearance and width rules.

selected

Applies routing rules to only the selected nets.

Use the **rule** command to set design rules for routing. Rules you set in SPECCTRA override rules set in the design file. See rules overview for general information about routing rules.

The object keyword determines the rule precedence level of the rules. For a list of the types of rules that apply to each rule precedence level, see routing rule hierarchy. Use <*routing_rules*> to set your rules.

You can use the **selected** keyword to apply rules to selected nets, but not to selected fromtos. To add or change fromto rules use the define net or define group commands.

For class-to-class rules, at least two class ID entries must be supplied. You can

- Apply rules between classes by listing multiple classes, where all classes are paired with each other.
- Apply rules between specific classes by listing only the two classes to be paired. You can enter the same class ID twice if you want to apply rules between the nets of a class.
- Apply parallel noise and tandem noise rules between two classes by listing only the two classes to be paired. The directional keyword determines which class is noise transmitter (first class specified) or noise receiver (second class specified). The **directional** keyword is used only for the <*parallel_noise_descriptor*> and the <*tandem_noise_descriptor*>.

Rules assigned to a region that have the same coordinates and layer range as an existing region are merged. Overlapping regions are allowed, but if rules conflict, the rules of the last defined region are used.

See also the define class_class and define region commands.

Tip

You can se a rule with the **define** command. For example:

To specify a width rule in the **rule** command

```
rule class class1 (width 600)
```

To specify a width rule in the **define** command

```
define (class class1 (sig1 sig2 sig3) (rule (width 600)))
```

Command examples

Click the button () by each example to go to a detailed description and syntax diagram.

This example sets a limit vias rule for each connection in the design.

```
rule pcb (limit_vias 3)
```

This example sets a clearance rule for a layer.

```
rule layer S1 (clearance 50 (type smd_to_turn_gap))
```

This example sets a parallel segment rule for a class.

```
rule class critical (parallel_segment (gap 25) (limit 150))
```

This example sets a parallel segment rule for a group.

```
rule group g1 (parallel_segment (gap 25) (limit 150))
```

This example sets a limit way rule for selected nets.

```
rule selected (limit_way 5)
```

This example sets a via at smd (**via_at_smd**) rule for the design.

```
rule pcb (via_at_smd on (grid on) (fit on))
```

This example sets noise rules for a class.

```
rule class clock (max_noise 400)
rule class clock (parallel_noise (gap 5) (threshold 50) (weight .04))
rule class clock (tandem_noise (gap 12) (threshold 50) (weight .01))
```

This example sets delay rules for a class.

```
rule class clock (time_length_factor .51)
circuit class clock (min_total_delay 1.2)
circuit class clock (max_total_delay 1.5)
```

This example sets a width rule for a region.

```
rule region region1 (width 10)
```

This example sets a junction type (**junction_type**) rule for a group set.

```
rule group_set grpset1 (junction_type term_only)
```

This example sets clearance rules for padstacks.

```
rule padstack V25 (clearance 20 (type via_via))
rule padstack V35 (clearance 25 (type via_via))
```

This example sets a pin width taper rule for a net.

```
rule net wr7 (pin_width_taper up_down)
```

This example sets a test point rule for all nets in the design.

```
rule pcb (testpoint (insert on) (grid 100)
```

This example sets an interlayer clearance rule between classes.

```
rule class_class C1 C2 (inter_layer_clear 3 (type wire_wire wire_pin)
  (layer_depth 2))
```

This example sets shielding rules for a net.

```
rule net wr9 (shield_gap 10)
rule net wr9 (shield_width 12)
rule net wr9 (shield_loop open)
```

This example sets a power fanout rule for all power pins in the design. The fanout

command will attempt to connect power pins to decoupling capacitors before escaping to a via.

rule pcb (power_fanout (pin_cap_via)

Rules overview

You can use the rule command to do the following:

- Set clearances
- Set interlayer clearances

- Set wire widths
- Control the width of the wire segment entering or exiting a pin

- Set a time conversion factor for wire delay.
- Set a factor to calculate effective wire length by layer and effective via length
- Set the maximum wire length permitted on a mixed layer
- Set a rule that marks a layer as restricted for routing .
- Set the length amplitude and length gap in accordion pattern routing

- Set net ordering to starburst or daisy
- Control use of tjunctions for starburst connections
- Control the maximum stub length and use of tjunctions for daisy chain connections
- Control tjunction types used in starburst and daisy chain routing

- Set the maximum number of bends permitted in a connection
- Set the maximum number of crossing conflicts permitted in a connection
- Set the maximum number of vias permitted in a connection and vias permitted in a net
- Set the maximum wrong-way distance permitted in a connection
- Permit interactive creation of redundant wiring on a net

- Set the maximum noise permitted on a net
- Control parallel noise and tandem noise calculations
- Control parallel segments and tandem segments for crosstalk considerations
- Set the minimum length beyond which the effect of noise saturation becomes a factor in noise calculations

- Control shield gap, shield wire width, and whether shield wire end loops are formed
- Control the amount by which tandem shields exceed the width of a shielded wire
- Control the distance between shield stub wires

- Control test point insertion during autorouting

- Control power pin fanout order to decoupling capacitors.

- Control the use of vias under SMD pads
- Set the via insertion pattern for multi-chip module design to spiral, staggered, or staired.

<allow_redundant_wiring_descriptor>

The *<allow_redundant_wiring_descriptor>* sets a rule that allows or disallows redundant wiring for a net during interactive routing.

When a net has an allow_redundant_wiring rule set to on, and redundant wiring is enabled in the interactive routing setup, the interactive router can create and leave wiring loops in the finished connection.

The default for this rule is off.

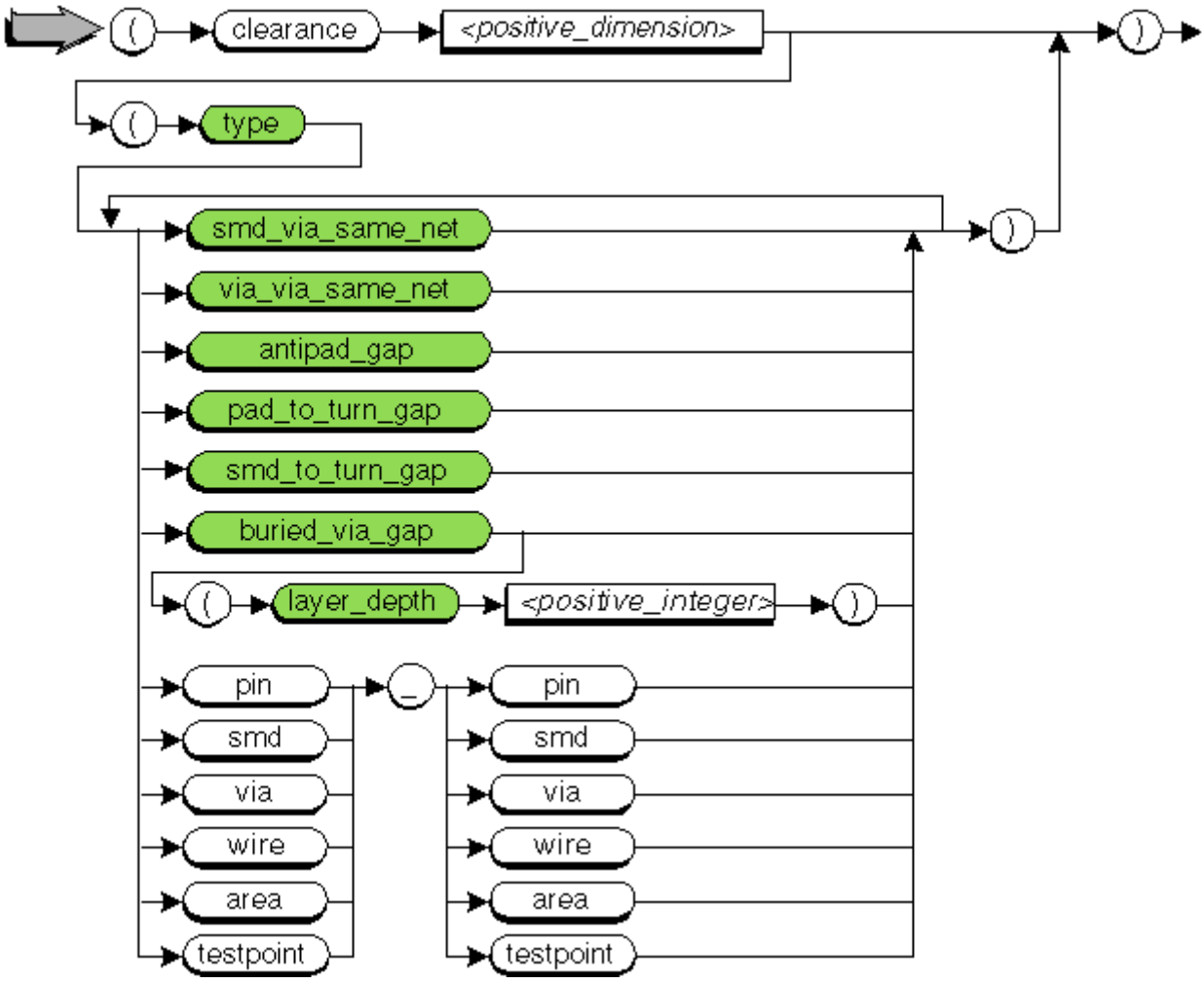
Notes

This rule is in effect only for interactive routing.

An allow_redundant_wiring rule on a net with daisy ordering is ignored.

<clearance_descriptor>

The *<clearance_descriptor>* sets a rule that controls clearances between routing objects in your design.



type

Identifies the objects to which you assign a clearance rule.

You can use the special clearance type keywords to specify clearance rules between objects such as vias attached to the same net (**via_via_same_net**).

You can use object-to-object clearance type keywords to specify clearances between two types of objects. The choices are through-pins (**pin**), SMD pads (**smd**), vias (**via**), routed trace segments (**wire**), keepout areas or the PCB routing boundary (**area**), and test points (**testpoint**). You must specify any combination of two of these keywords with an underscore between them.

For example

(type pin_pin)
(type pin_wire)

If you do not specify **type**, clearance rules apply between all object types.

smd_via_same_net

An SMD pad and a via attached to the same net.

via_via_same_net

Vias attached to the same net.

antipad_gap

A pin and the surrounding copper plane.

pad_to_turn_gap

The edge of a pin and the first turn or bend in the wire segment.

smd_to_turn_gap

An SMD pad and the first turn or bend in the wire segment.

buried_via_gap

The gap between vias on different layers. You can use the **layer_depth** option to control how many adjacent layers are checked for buried via clearance.

layer_depth

Controls how many adjacent layers (<*positive_integer*>) are considered when the autorouter checks buried via clearance.

If the number of layers between vias is larger than specified by <*positive_integer*>, the clearance rule does not apply.

Use the **clearance** rule to set the minimum distance (<*positive_dimension*>) permitted between routing objects within the PCB outline.

A value of **0** means the edges of objects can meet. A value of **-1** means the rule is not specified.

Use **type** to identify the objects to which you assign a clearance rule. You can identify

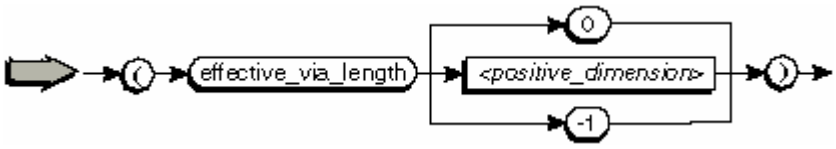
- Special clearance types with one keyword
- Object-to-object clearance types with separate keywords that are joined by an underscore character

If you do not specify **type**, the clearance rule applies to all object types.

For **buried_via_gap**, you can control how many adjacent layers are checked when you specify the gap between vias on different layers.

<effective_via_length_descriptor>

The <*effective_via_length_descriptor*> sets a rule that controls the amount that is added to wire length calculations by through-vias.

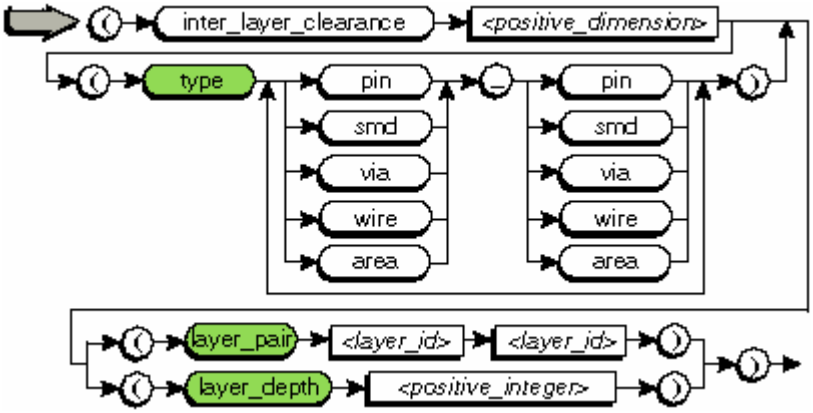


Use the **effective_via_length** rule to set the amount (<*positive_dimension*>) that is added to wire length calculations by each through-via.

A value of **0** means length is not added per via. A value of **-1** turns off the rule.

<inter_layer_clearance_descriptor>

The <*inter_layer_clearance_descriptor*> sets a rule that controls clearances between objects on different layers.



type

Identifies the objects to which you assign an interlayer clearance rule.

You can use object-to-object keywords to specify clearances between two types of objects. The choices are through-pins (**pin**), SMD pads (**smd**), vias (**via**), routed wire segments (**wire**), and keepout areas and the PCB routing boundary (**area**). You can use any combination of two object keywords separated by an underscore character.

For example

(type pin_pin)
(type pin_wire)

If you do not specify **type**, clearance rules apply to all object types.

layer_pair

Controls the layer pair between which interlayer clearance rules apply. You must specify a layer name (<*layer_id*>) for each layer in the pair.

The **layer_pair** control applies at the pcb (global) level of the rule hierarchy only.

layer_depth

Identifies how many adjacent layers (<*positive_integer*>) are considered when the

autorouter applies interlayer clearance rules between objects in one class and those of another.

The **layer_depth** control applies at the class-to-class level of the rule hierarchy only.

Use the **inter_layer_clearance** rule to set the minimum distance (<*positive_dimension*>) permitted between objects that do not occupy the same layer. You identify the objects by using **type** and object-to-object keywords. If you do not use **type**, the interlayer clearance rules apply to all object types.

You control which layers rules apply to by using **layer_pair** to specify a pair of layers at the pcb level and **layer_depth** to the number of adjacent layers at the class-to-class level. See rule hierarchy for information about rule precedence.

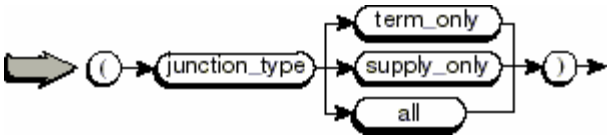
Notes

A common use for interlayer clearance rules is to keep digital nets from crossing analog nets. You define a class for analog nets and a class for digital nets and assign a class-to-class interlayer clearance rule with **layer_depth** control. For example

```
rule class_class C1 C2 (inter_layer_clear 3 (type wire_wire wire_pin)
   (layer_depth 2))
```

<*junction_type_descriptor*>

The <*junction_type_descriptor*> sets a rule that controls whether tjunctions occur at pins, pads, vias, and at wire segments.



Use **junction_type** to control where tjunctions occur. The choices are

term_only, which permits tjunctions at pins, pads, and vias.

supply_only, which permits tjunctions only at pins and pads connected to source-terminals.

all, which permits tjunctions at pins, pads, and vias, and on wire segments.

For starburst routing, the tjunction rule must be turned on before you use **junction_type**. For daisy chain routing, the max_stub rule must be set to a value greater than 0 before you use **junction_type**.

Note

Individual pins, wires, and wiring polygons can be defined as source-terminals with the assign_supply command.

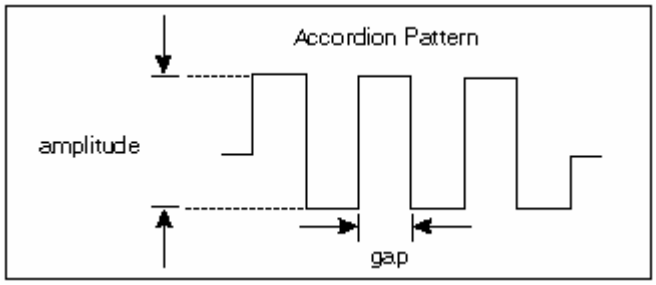
<length_amplitude_descriptor>

The <*length_amplitude_descriptor*> sets a rule that controls the amplitude permitted in accordion pattern routing.

The **length_amplitude** rule controls the amplitude permitted in accordion pattern routing that occurs when wire is added to satisfy a min_length rule. <*max_amp*> is a positive_dimension value that sets the maximum amplitude. <*min_amp*> is an optional positive_dimension value that sets the minimum amplitude. If a value is not specified for <*min_amp*>, the minimum length amplitude defaults to the greater of three times the wire width or one wire width plus one wire-wire clearance.

A value of **0** for <*max_amp*> prevents the accordion pattern and forces maze routing to satisfy a **min_length** rule. A value of **-1** for <*max_amp*> turns off the accordion pattern. A value of **-1** for <*min_amp*> returns minimum length amplitude to the default value.

An example of an accordion pattern is shown in the following figure. See the length_gap rule for information about controlling distance between accordion segments.



<length_factor_descriptor>

The <*length_factor_descriptor*> sets a rule that defines the factor for calculating the effective length of wires on a layer.

Use **length_factor** to set a multiplier (<*real*>) used to calculate the effective length of wires on a layer. This value must be equal to or greater than 0.

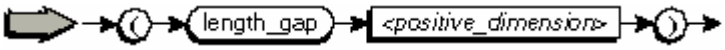
A value of **-1** sets the rule to unspecified.

The length factor adjusts wire length calculations by layer. Actual wire lengths are multiplied by a length factor to derive the effective routed length on a layer.

<length_gap_descriptor>

The <*length_gap_descriptor*> sets a rule that defines the gap permitted between

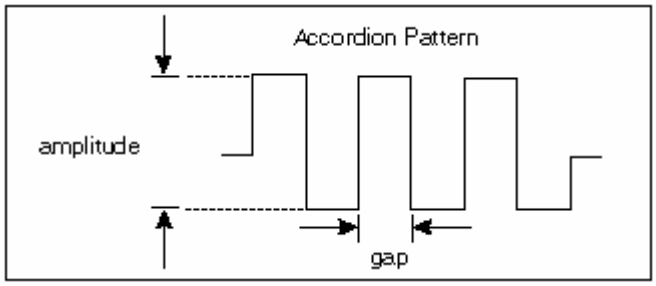
adjacent folded segments in accordion pattern routing.



The **length_gap** rule controls the distance or gap (<*positive_dimension*>) between adjacent folded segments when wire is added to satisfy a min_length rule with accordion pattern routing.

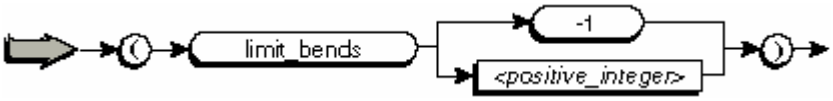
This rule is ignored if the <*positive_dimension*> is equal to or less than the wire-to-wire clearance rule for the wire segment.

An example of an accordion pattern is shown in the following figure. See the length_amplitude rule for information about controlling the amplitude permitted in accordion routing patterns.



<*limit_bends_descriptor*>

The <*limit_bends_descriptor*> sets a rule that defines the maximum number of bends permitted in a connection.

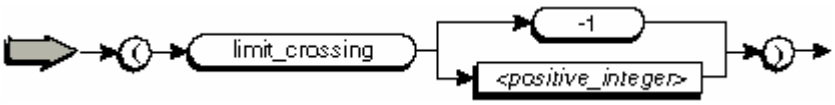


Use the **limit_bends** rule to control the maximum number (<*positive_integer*>) of bends used to route a connection. The positive integer must be a value from 0 to 255.

A value of **-1** sets the rule to unspecified, which means the autorouter calculates the maximum number of bends internally.

<*limit_crossing_descriptor*>

The <*limit_crossing_descriptor*> sets a rule that defines the maximum number of crossing conflicts permitted in a connection.

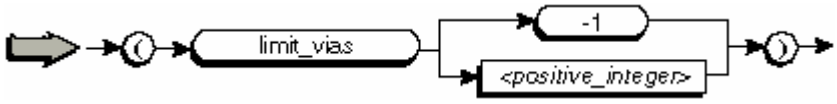


Use the **limit_crossing** rule to control the maximum number (<*positive_integer*>) of crossing conflicts that are allowed to route a connection. The positive integer must be a value from 0 to 255.

A value of **-1** sets the rule to unspecified, which means the autorouter calculates the maximum number of crossing conflicts internally.

<limit_vias_descriptor>

The <*limit_vias_descriptor*> sets a rule that defines the maximum number of vias permitted in a connection.



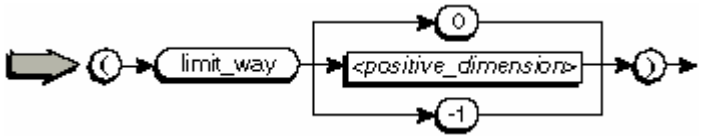
Use the **limit_vias** rule to control the maximum number (<*positive_integer*>) of vias used to route a connection. The positive integer must be a value from 0 to 255.

A value of **-1** sets the rule to unspecified, which means the autorouter calculates the maximum number of vias internally.

This rule controls the number of vias used in a fromto. See also the max_total_vias rule for information about controlling the number of vias in a net.

<limit_way_descriptor>

The <*limit_way_descriptor*> sets a rule that defines the maximum wrong-way distance permitted in a connection.



Use the **limit_way** rule to limit the maximum wrong-way distance (<*positive_dimension*>) permitted when a connection is routed. The positive dimension must be correctly scaled for your current measurement units.

A value of **0** prevents any wrong-way routing. A value of **-1** sets the rule to unspecified, which means the autorouter calculates the wrong-way distance internally.

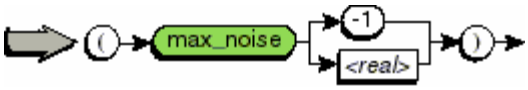
The wrong-way direction is vertical on horizontal routing layers and horizontal on vertical routing layers.

Notes

A value of **0** can significantly increase the total number of vias in the design.

<max_noise_descriptor>

The <*max_noise_descriptor*> sets a rule that controls the maximum noise permitted on a net.



max_noise

Controls the total noise allowed to accumulate on a net.

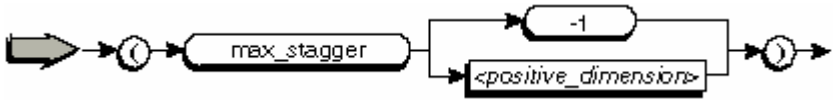
Use the **max_noise** rule to identify the maximum noise (<*real*>) that can accumulate on a net before a coupled noise violation occurs.

A value of **-1** sets the rule to unspecified.

When violations occur, the wires involved in the calculations are rerouted to reduce the noise below the minimum value. See also the parallel_noise and tandem_noise rules.

<*max_stagger_descriptor*>

The <*max_stagger_descriptor*> sets a rule that controls the maximum wire length permitted on a mixed layer.



Use the **max_stagger** rule to set the maximum wire length (<*positive_dimension*>) permitted on a mixed layer. The value must be correctly scaled for your current measurement units.

A value of **-1** sets the rule to unspecified, and therefore a connection can be routed without length restrictions on a mixed layer.

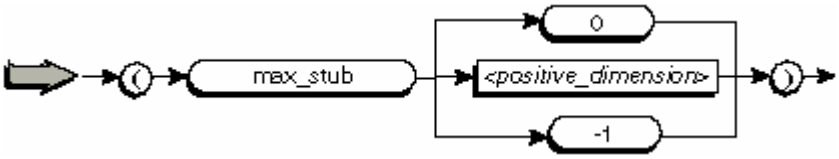
A mixed layer is a power layer that can also be used to route signal connections.

Note

The **max_stagger** rule should be set at the layer, class layer, net layer, fromto layer, and group layer levels only.

<*max_stub_descriptor*>

The <*max_stub_descriptor*> sets a rule that controls the maximum stub length for daisy chain connections.



Use the **max_stub** rule to set the maximum stub length (<*positive_dimension*>) allowed on daisy chain connections

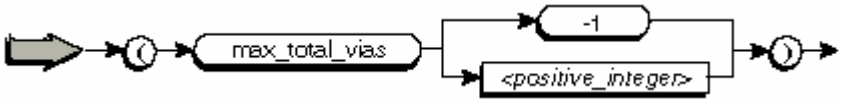
Stub length is the distance between a pin or via and a tjunction. Stub length is measured from the center of a pad to the center of the tjunction.

A value of **0** prevents stubs. A value of **-1** resets the rule to unspecified.

A stub length greater than 0 permits tjunctions on daisy chain connections. You can use junction_type to control whether tjunctions can occur on wires or only at pins, pads, and vias.

<max_total_vias_descriptor>

The <*max_total_vias_descriptor*> sets a rule that controls the maximum number of vias permitted in a net.



Use the **max_total_vias** rule to set the maximum number (<*positive_integer*>) of vias that are used to route the net.

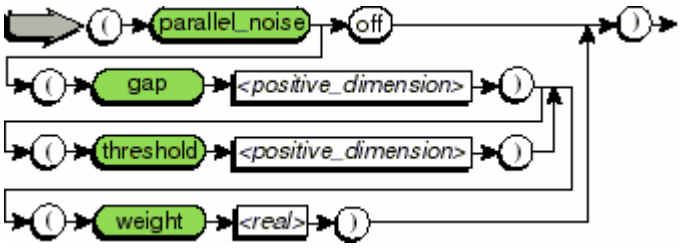
A value of **-1** sets the rule to unspecified.

You can set this rule for a net, for fromtos in a group, or for nets in a class. If applied to a class, the rule limits the maximum number of vias for each net in the class, not the total number of vias for the class.

See also the limit_vias rule for information about controlling the number of vias in a fromto.

<parallel_noise_descriptor>

The <*parallel_noise_descriptor*> sets a rule that controls noise calculations between parallel wires on the same layer.



parallel_noise

Controls whether parallel coupled noise is calculated for parallel wires on the same layer. The choices are

off turns off the **parallel_noise** rule, which means parallel coupled noise is not calculated.

gap turns on the **parallel_noise** rule and sets the **gap**, **weight**, and **threshold** values used to calculate parallel coupled noise.

Turn off **parallel_noise** before you set new parallel noise rules.

gap

Sets the edge-to-edge distance (<*dimension*>) at which parallel or tandem coupled noise calculations are made.

Coupled noise is calculated for parallel or tandem wires when the edge-to-edge distance is equal to or less than the specified **gap** value and the wires are parallel for a distance that exceeds the **threshold** value. A negative value for <*dimension*> implies overlapping wires.

threshold

Sets the minimum distance (<*positive_dimension*>) above which parallel wires are included in parallel or tandem noise calculations.

Coupled noise is calculated for parallel or tandem wires when the wires are parallel over a distance that exceeds the **threshold** value, and the edge-to-edge distance is equal to or less than the specified **gap** value.

If **threshold** is not set, the **gap** value is used for threshold.

weight

Sets the noise transmitted by a net per unit of routed wire length. The noise **weight** is used in the cct1 crosstalk model. The value (<*real*>) must be in electrical units consistent with the dimensional unit set in SPECCTRA. For example, if coupling between parallel wires is 2 millivolts per millimeter, **weight** is set as 2.

Coupled noise is calculated by multiplying parallel lengths by the **weight** value of the transmitting net.

Use the **parallel_noise** rule to control how SPECCTRA calculates coupled noise specifications between parallel wires on the same layer.

To control coupled noise, you set an edge-to-edge distance (**gap**) between parallel wires and a noise weight (**weight**). The noise **weight** is used in the cct1 crosstalk model.

You can also set an optional parallel wire length threshold (**threshold**). Multiple gap, threshold, and weight rules can be set to approximate a noise coupling characteristic that varies as a function of gap and length.

The total accumulated noise on a victim net is compared to **Max Noise**. Depending on the setting of the noise accumulation parameter in the set command, this total is calculated as a linear sum or as the square root of the sum of squares of the noise contributions of the aggressor nets. The default setting is linear.

SPECCTRA calculates the total noise coupled to the victim net from parallel transmitting wires by multiplying the parallel length by the weight of each transmitting wire and accumulating all coupled noise contributions. Depending on the setting of the noise accumulation parameter in the set command, this total is calculated as a linear sum or as the square root of the sum of squares of the noise contributions. (The default setting is linear.) The sum is compared with the net's maximum noise

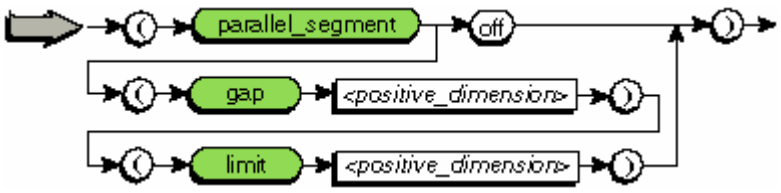
specification to determine if a violation exists.

Use the max_noise rule to set the maximum noise that each receiving net can tolerate. When the total coupled noise exceeds the **max_noise** rule for the net, the condition is a violation and SPECCTRA reroutes the net to comply with the coupled noise rule.

See also the tandem_noise rule to control noise coupling between wires on adjacent signal layers. You can use the parallel_segment rule to control crosstalk by limiting segments of wire length for a given gap on the same layer. Use the tandem_segment rule to control crosstalk by limiting segments of wire length for a given gap on adjacent layers.

<parallel_segment_descriptor>

The <*parallel_segment_descriptor*> sets a rule that controls segment crosstalk between nets routed on the same layer.



parallel_segment

Controls whether parallel crosstalk is considered for parallel wire segments on the same layer. The choices are

off turns off the **parallel_segment** rule, which means parallel segment crosstalk is not considered.

gap turns on the **parallel_segment** rule and sets the **gap** and **limit** values used to consider parallel segment crosstalk.

Turn off **parallel_segment** before you specify new parallel segment rules.

gap

Sets the minimum edge-to-edge distance, or gap (<*dimension*>), between parallel wire segments at which parallel or tandem segment violations occur.

The violations occur when the edge-to-edge distance is equal to or less than the specified **gap** value.

When parallel wires are separated by a distance that is less than the **gap** value, and the wires are parallel for a distance that exceeds the length **limit** value, the wires are rerouted during subsequent routing passes to correct the condition. A negative value for <*dimension*> implies overlapping wires.

limit

Sets the maximum distance (<*positive_dimension*>) that wire segments can be parallel before a violation can occur.

Wire segment lengths equal to or less than the **limit** value are not considered in parallel segment and tandem segment violations.

When wires are parallel for a distance that exceeds the length **limit** value, and the edge-to-edge distance is equal to or less than the specified **gap** value, the wires are rerouted during subsequent routing passes to correct the condition.

Use the **parallel_segment** rule to control crosstalk between nets routed on the same layer by limiting the distance wire segments are routed in parallel at a given gap.

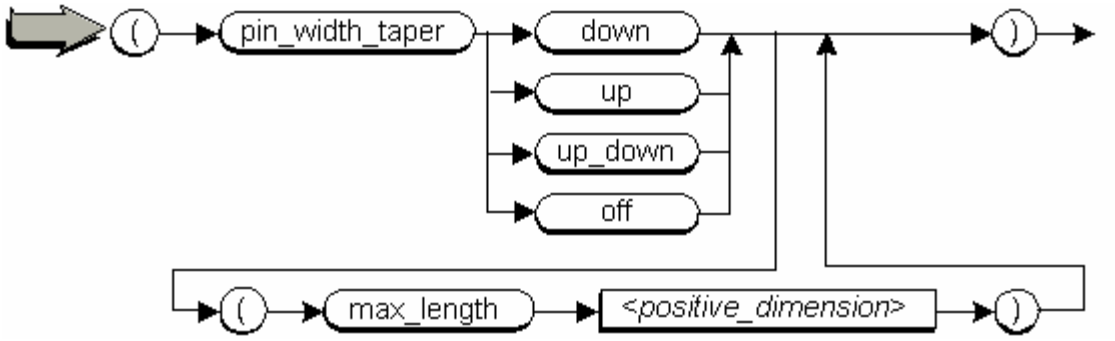
To prevent parallel segment violations, you set an edge-to-edge distance (**gap**) and a parallel segment length limit (**limit**). You can set different parallel length limits for different gaps by using multiple **parallel_segment** rules.

These rules apply only to individual wire segments and are not cumulative. To route a net so that the total noise on the net does not exceed a specified limit, see the parallel_noise rule.

See also the tandem_segment and tandem_noise rules for information about segment control and noise control between wires on adjacent signal layers.

<pin_width_taper_descriptor>

The *<pin_width_taper_descriptor>* sets a rule that controls the width of a wire segment entering or exiting a pin.



Use the **pin_width_taper** rule to control the width of the wire segment entering or exiting a pin so that it matches the width of the pin or equals the pcb width rule. The choices are:

down, which reduces the wire segment.

up, which enlarges the wire segment if no violation to adjacent pins occurs.

up_down, which reduces or enlarges the wire segment as needed.

off, which turns off pin width tapering.

max_length, which limits the length of the tapered portion of the wire.

The default is **down**.

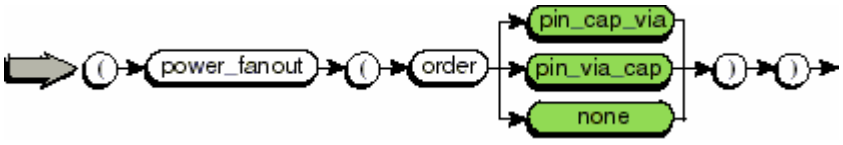
The **pin_width_taper** rule matches the connecting segment width of a wire to the pin width, when wire and pin widths differ. All other segments of the wire obey the width

rule that applies to the wire as a whole. No width tapering occurs if it leads to any rule violations.

The **max_length** option permits a tapered wire portion that is shorter than the length of the wire segment connecting to the pin. If you specify a **max_length** value that is longer than the connecting wire segment, only the segment entering the pin is tapered.

<power_fanout_descriptor>

The *<power_fanout_descriptor>* sets a rule that specifies the fanout routing order between power pins, vias, and decoupling capacitors.



pin_cap_via

Sets the fanout routing order to fanout from a pin directly to a bypass capacitor before a via.

pin_via_cap

Sets the fanout routing order to fanout from a pin directly to a via before a bypass capacitor.

none

Removes the power fanout routing order rule.

Use this rule to control the order in which fanout connects power pins of large components to decoupling capacitors and vias at the PCB, NET, and CLASS levels. The rule sets the order to pin-via-cap or pin-cap-via, or removes an existing power fanout order.

The rule applies only to power nets and to components that are categorized as follows:

A "large" component must have at least four pins and must have a component type property of "large".

A decoupling capacitor must have two pins with one pin connected to a voltage source and the other to ground, and must have a component type property of "capacitor".

Notes

Power fanout order violations occur when the router cannot follow the order specified in the rule. You can use the **highlight** and **report** commands to discover power fanout order violations.

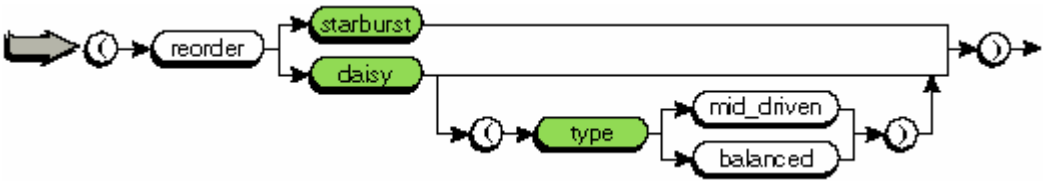
See also

highlight command

report command

<reorder_descriptor>

The <*reorder_descriptor*> sets a rule that defines net ordering as starburst or daisy.



starburst

Permits multiple entries and exits on pins.

daisy

Permits only a single entry and a single exit on each pin in the net and does not allow tjunctions. This is called a simple daisy chain. You can choose mid-driven or balanced daisy chain routing by using the **type** option.

type

Controls how a net is ordered for daisy chain routing. The choices are:

mid_driven, where a terminator is placed at each end of the net, and the loads are added back to a source. If there is more than one source, the sources are chained together first before the rest of the net is processed.

balanced, where fromtos are daisy-chained and loads are equally distributed between source and terminator pins. If more than one source pin is defined, the terminator and load branches are chained back to the closest source pin and the remaining source pins are ordered as simple daisy chain.

Use the reorder rule to control which method of ordering fromtos in nets is used. Choose **starburst** when multiple entries and exits on pins are permitted in your design (the best routing results are obtained with starburst routing). Choose **daisy** if your design requires single exit and entry on pins, and no tjunctions.

When you choose **daisy**, the net is ordered as a simple daisy chain. You can choose a **type** option to control how a net is ordered for daisy chain routing. If you choose **mid_driven**, there must be exactly two terminator pins and one or more source pins. If you choose **balanced**, loads are equally distributed between source and terminator pins.

You can control tjunctions in your starburst and daisy chain routing

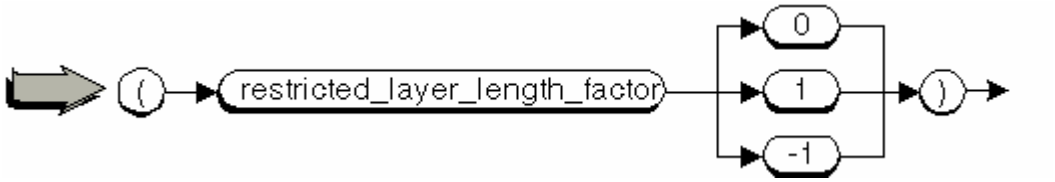
- For starburst routing, tjunctions are permitted if the tjunction rule is on. You can

use the junction_type rule to control whether tjunctions occur at only pins, pads, and vias, or on wire segments too.

- For daisy chain routing, tjunctions are permitted if the max_stub rule is set to a value greater than 0. You can use the junction_type rule to control whether tjunctions occur at only pins, pads, and vias, or on wire segments too.

<restricted_layer_length_factor_descriptor>

The <*restricted_layer_length_factor_descriptor*> sets a rule that marks a layer as restricted for routing.



Use the **restricted_layer_length_factor rule** at the layer, class_layer, or net_layer precedence level to restrict routing on certain layers for all nets, nets in a certain class, or specific individual nets, respectively. The rule acts as a switch to identify layers as restricted. A value of 1 marks a layer as restricted. A value of 0 removes restrictions from a layer. A value of -1 sets layer restrictions to unspecified. By default, all layers have a restricted layer length factor of 0.

For example,

```
rule layer sig1 sig4 (restricted_layer_length_factor 1)
```

marks layers sig1 and sig4 as restricted. Only nets with a restricted layer rule will be routed on those layers.

```
define (class restricted (selected) (layer_rule sig1sig4 (rule /
(restricted_layer_length_factor 1)))
```

marks the layers as restricted at the class_layer level, meaning that routing restrictions apply to nets in the class "restricted" on those layers.

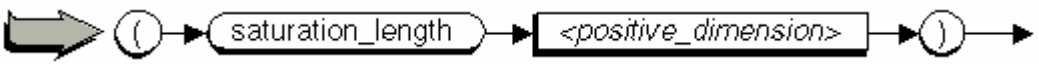
Note

Routing on a restricted layer is limited to nets with a restricted layer circuit rule. Restricted layer circuit rules include the following:

max_restricted_layer_length

<saturation_length_descriptor>

The <*saturation_length_descriptor*> sets the minimum length beyond which the effect of noise saturation becomes a factor in noise calculations.



The **saturation_length** rule sets a value for saturation length that is included in noise calculations. When the total parallel length of a victim and aggressor pair exceeds the saturation length, the noise calculation scales the total noise by the ratio of the saturation length to the total parallel length.

This rule applies to parallel and tandem noise calculations at the pcb, class, and net levels of the rule hierarchy when the cct1a crosstalk model is in use.

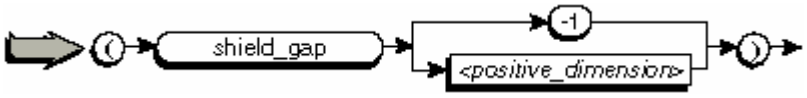
Note

SPECCTRA uses the cct1 crosstalk model by default. To make use of the saturation_length rule, the cct1a crosstalk model must be set. This can occur in either of two ways:

- in the design file by use of the **crosstalk_model** keyword in the <*control_descriptor*>, (see the *SPECCTRA Design Language Reference*) or
- by using the **set crosstalk_model cct1a** command

<*shield_gap_descriptor*>

The <*shield_gap_descriptor*> sets a rule that controls the gap between a shield wire and the signal wires that are being shielded.

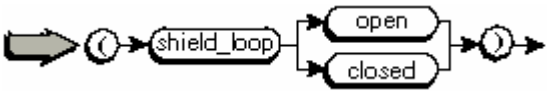


Use the **shield_gap** rule to control the edge-to-edge distance or gap (<*positive_dimension*>) permitted between the shield wire and the signal wires being shielded.

A specified shield_gap takes precedence over an existing wire-to-wire clearance value. A value of **-1** sets the rule to unspecified, and the gap is determined by the wire-to-wire clearance rule for the signal wires that are being shielded.

<*shield_loop_descriptor*>

The <*shield_loop_descriptor*> sets a rule that controls whether shield wires meet in a closed end loop.



Use the **shield_loop** rule to control whether shield wire ends meet to enclose the signal wire they are shielding. The choices are

closed, which means SPECCTRA routes shield wiring with closed end loops

open, which means SPECCTRA routes shield wiring without closing the ends

When using **open**, SPECCTRA usually adds a via to each of the two shield wires to connect them to the assigned power layer.

The default is **closed**.

<shield_tie_down_interval_descriptor>

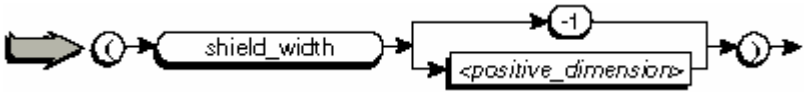
The <shield_tie_down_interval_descriptor> sets a rule that controls the distance between shield stub wires.

Use the **shield_tie_down_interval** rule to set the distance (<*positive_dimension*>) between stub wires that connect a shield to the ground plane.

A value of **-1** sets the rule to unspecified.

<shield_width_descriptor>

The <shield_width_descriptor> sets a rule that controls shield wire width.



Use the **shield_width** rule to set the width (<*positive_dimension*>) of the shield wire.

A value of **-1** sets the rule to unspecified, and the width is determined by the same width as the signal wires being shielded.

<spiral_via_descriptor>

The *<spiral_via_descriptor>* sets a rule that controls autorouter insertion of spiral via patterns.

min_gap

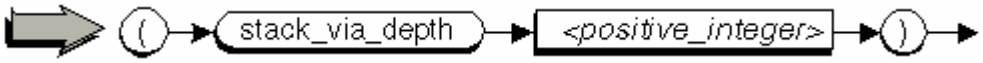
Sets the minimum horizontal distance between vias in the same pattern on adjacent layers.

Use the *<spiral_via_descriptor>* with the **rule** command to set rules at the PCB, layer, class, net, group, group set, and fromto levels. This rule is **off** by default. When the rule is **on**, **min_gap** controls the minimum distance between consecutive vias in the pattern. If **min_gap** is not specified, the largest via_via clearance rule in effect controls the distance on all layers of the pattern.

The autorouter connects each via in the pattern at a right angle to the previous via, resulting in a pattern of vias and connections that form a square if viewed from above.

<stack_via_depth_descriptor>

The <*stack_via_depth_descriptor*> sets a rule that controls the layer span of stacked vias.

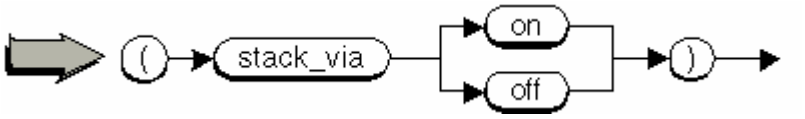


Use the **stack_via_depth** rule to control the layer span of stacked blind/buried vias. The rule applies only at the PCB level of the rule hierarchy and works in conjunction with the stack_via rule, which enables via stacking. For example, to turn on via stacking and allow a layer span of 3 layers for stacked vias, you could enter the following commands.

```
rule pcb (stack_via on)
rule pcb (stack_via_depth 3)
```

<stack_via_descriptor>

The <*stack_via_descriptor*> sets a rule that controls center-on-center via stacking.



Use the **stack_via** rule to control via stacking. The choices are

on turns on **stack_via**, which means two vias can be stacked if the terminal points of the two vias are the same, resulting in a center-to-center stackup.

off turns off **stack_via**, which means vias cannot be stacked.

The **stack_via** rule applies at the PCB and layer precedence levels of the rule hierarchy. For example, to allow overlapping vias in a design, you could enter the following.

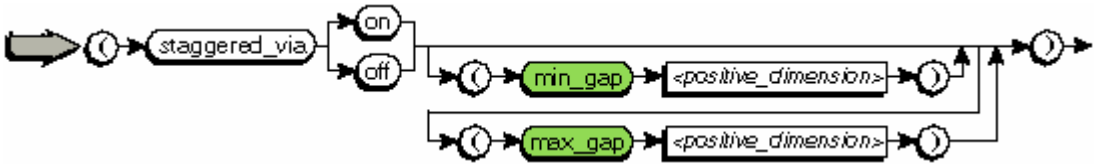
```
rule pcb (stack_via on)
```

Note

The stack_via rule enables via stacking. When used in conjunction the stack_via_depth rule, you can place blind and buried vias at the same location on different layers.

<staggered_via_descriptor>

The *<staggered_via_descriptor>* sets a rule that controls autorouter insertion of staggered via patterns.



min_gap

Sets the minimum horizontal distance between vias in the same pattern on adjacent layers.

max_gap

Sets the maximum horizontal distance between vias in the same pattern on adjacent layers.

Use the *<staggered_via_descriptor>* with the **rule** command to set rules at the PCB, layer, class, net, group, group set, and fromto levels. This rule is **off** by default. When the rule is **on**, **min_gap** controls the minimum distance between consecutive vias in the pattern. If **min_gap** is not specified, the largest via_via clearance rule in effect controls the distance on all layers of the pattern.

The autorouter connects each via in the pattern at a 180 degree angle to the previous via, resulting in a pattern of vias and connections that form a straight line that doubles back on itself after each via connection.

<staired_via_descriptor>

The *<staired_via_descriptor>* sets a rule that controls autorouter insertion of staired via patterns.

min_gap

Sets the minimum horizontal distance between vias in the same pattern on adjacent layers.

max_gap

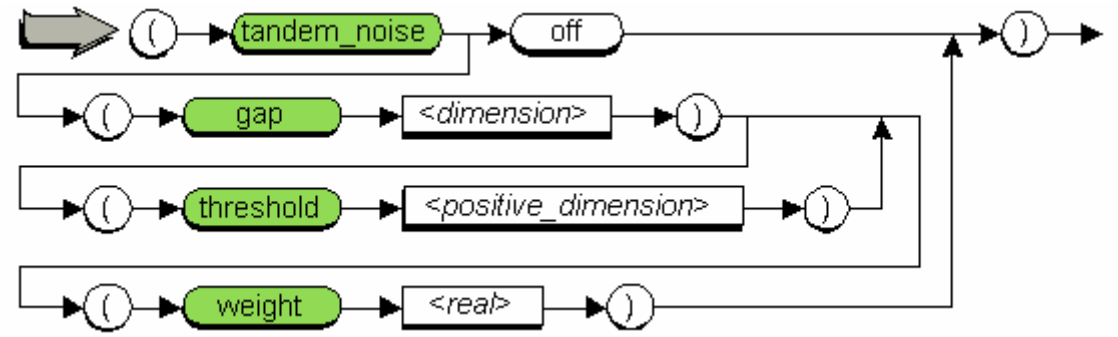
Sets the maximum horizontal distance between vias in the same pattern on adjacent layers.

Use the *<staired_via_descriptor>* with the **rule** command to set rules at the PCB, layer, class, net, group, group set, and fromto levels. This rule is **off** by default. When the rule is **on**, **min_gap** controls the minimum distance between consecutive vias in the pattern. If **min_gap** is not specified, the largest via_via clearance rule in effect controls the distance on all layers of the pattern.

The autorouter proceeds in a single direction to connect each via in the pattern, resulting in a pattern of vias and connections that form a straight line.

<tandem_noise_descriptor>

The <*tandem_noise_descriptor*> sets a rule that controls noise calculations between parallel wires on adjacent signal layers.



tandem_noise

Controls whether parallel coupled noise is calculated for parallel wires on adjacent signal layers. The choices are

off turns off the **tandem_noise** rule, which means parallel coupled noise is not calculated.

gap turns on the **tandem_noise** rule, and sets the **gap**, **weight**, and **threshold** values used to calculate parallel coupled noise.

Use **tandem_noise off** before you specify new rules.

gap

Sets the edge-to-edge distance (<*dimension*>) at which parallel or tandem coupled noise calculations are made.

Coupled noise is calculated for parallel or tandem wires when the edge-to-edge distance is equal to or less than the specified **gap** value and the wires are parallel for a distance that exceeds the **threshold** value. A negative value for <*dimension*> implies overlapping wires.

threshold

Sets the minimum distance (<*positive_dimension*>) above which parallel wires are included in parallel or tandem noise calculations.

Coupled noise is calculated for parallel or tandem wires when the wires are parallel over a distance that exceeds the **threshold** value, and the edge-to-edge distance is equal to or less than the specified **gap** value.

If **threshold** is not set, the **gap** value is used for threshold.

weight

Sets the noise transmitted by a net per unit of routed wire length. The noise **weight** is used in the cct1 crosstalk model. The value (<*real*>) must be in electrical units consistent with the dimensional unit set in SPECCTRA. For example, if coupling between parallel wires is 2 millivolts per millimeter, **weight** is set as 2.

Coupled noise is calculated by multiplying parallel lengths by the **weight** value of the transmitting net.

Use the **tandem_noise** rule to control how SPECCTRA calculates parallel coupled noise between nets on adjacent signal layers.

To control coupled noise, you set an edge-to-edge distance (**gap**) between parallel wires and a noise weight (**weight**). The noise **weight** is used in the cct1 crosstalk model. .

You can also set an optional parallel wire length threshold (**threshold**). Multiple gap, threshold, and weight rules can be set to approximate a noise coupling characteristic that varies as a function of gap and length.

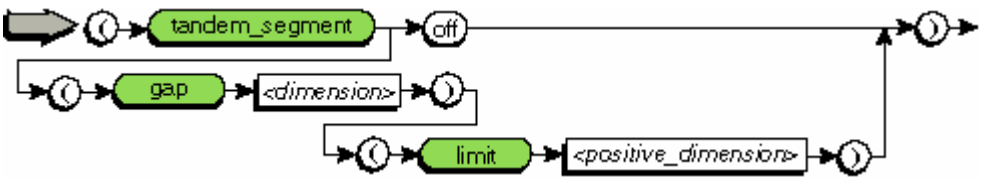
SPECCTRA calculates the total noise coupled to the victim net from tandem transmitting wires by multiplying the parallel length by the weight of each transmitting wire and accumulating all coupled noise contributions. Depending on the setting of the noise accumulation parameter in the set command, this total is calculated as a linear sum or as the square root of the sum of squares of the noise contributions. (The default setting is linear.) The sum is compared with the net's maximum noise specification to determine if a violation exists.

See the max_noise rule to set the maximum noise that each receiving net can tolerate. When the total coupled noise exceeds the **max_noise** rule for the net, the condition is a violation and SPECCTRA reroutes the net to comply with the coupled noise rule.

See also the parallel_noise rule to control noise coupling between wires on the same layer. You can use the parallel_segment rule to control crosstalk by limiting segments of wire length for a given gap on the same layer. Use the tandem_segment rule to control crosstalk by limiting segments of wire length for a given gap on adjacent layers.

<tandem_segment_descriptor>

The <*tandem_segment_descriptor*> sets a rule that specifies segment crosstalk control between nets routed on adjacent signal layers.



tandem_segment

Controls whether parallel crosstalk is considered for parallel wire segments on adjacent signal layers. The choices are

off turns off the **tandem_segment** rule, which means parallel segment crosstalk is not considered.

gap turns on the **tandem_segment** rule and sets the **gap** and **limit** values used to consider parallel segment crosstalk.

Turn off **tandem_segment** before you specify new tandem segment rules.

gap

Sets the minimum edge-to-edge distance, or gap (<*dimension*>), between parallel wire segments at which parallel or tandem segment violations occur.

The violations occur when the edge-to-edge distance is equal to or less than the specified **gap** value.

When parallel wires are separated by a distance that is less than the **gap** value, and the wires are parallel for a distance that exceeds the length **limit** value, the wires are rerouted during subsequent routing passes to correct the condition. A negative value for <*dimension*> implies overlapping wires.

limit

Sets the maximum distance (<*positive_dimension*>) that wire segments can be parallel before a violation can occur.

Wire segment lengths equal to or less than the **limit** value are not considered in parallel segment and tandem segment violations.

When wires are parallel for a distance that exceeds the length **limit** value, and the edge-to-edge distance is equal to or less than the specified **gap** value, the wires are rerouted during subsequent routing passes to correct the condition.

Use the **tandem_segment** rule to control crosstalk between nets routed on adjacent signal layers by limiting the lengths of parallel wire segments for a given gap.

To prevent parallel segment violations, you set an edge-to-edge distance (**gap**) and a parallel segment length limit (**limit**). You can set different parallel length limits for different gaps by using multiple **tandem_segment** rules.

These rules are applied only to individual wire segments and are not cumulative. To route a net so that the total noise on the net does not exceed a specified limit, see the tandem_noise rule.

See also the parallel_segment and parallel_noise rules for information about segment control and noise control between wires on the same layer.

<tandem_shield_overhang_descriptor>

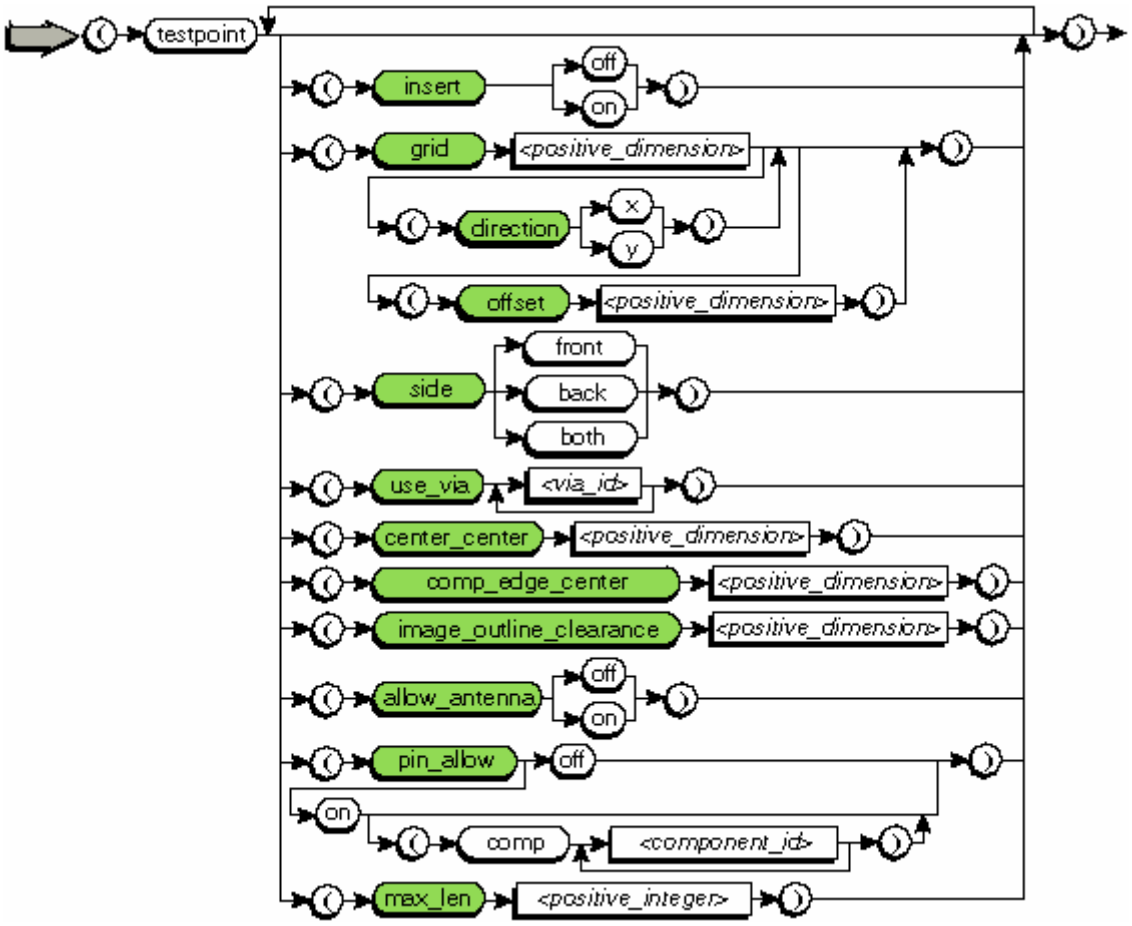
The <*tandem_shield_overhang_descriptor*> sets a rule that controls the width of the

shield wires generated when a net is routed with a circuit shield rule set to tandem.

Use this descriptor to specify the extra amount added to each side of the tandem shield wire. Total tandem shield width is two times the tandem_shield_overhang value plus the width of the wire being shielded. The tandem_shield_overhang value defaults to the width of the shield wire, resulting in a shield width three times the shielded wire width.

<testpoint_rule_descriptor>

The <*testpoint_rule_descriptor*> sets a rule that controls test point insertion during autorouting.



insert

Controls whether test points are added to routed signal nets. The choices are

on turns on **insert**, which means that SPECCTRA marks test points and inserts test point vias.

off turns off **insert,** which means that SPECCTRA does not mark test points and insert test point vias.

The default is **off**.

grid

Defines a uniform grid or nonuniform X and Y grids. Grids can be offset. You can

Specify the grid value (<*positive_dimension*>)

Specify an X or Y direction (**direction**)

Specify an offset (**offset**)

If you want a uniform grid, do not specify a direction.

The default test point grid is the current pcb via grid. The grid for test point insertion is a probing grid that should match your bed-of-nails tester.

direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

side

Identifies the test point probing layer as the top (**front**), bottom (**back)**, or both top and bottom (**both)** sides of the PCB.

The probing layer contains exposed test vias (not covered by a component body).

The default is **back**.

use_via

Identifies one or more via padstacks (<*via_id*>) to be used as test points.

If no **use_via** value is specified, the autorouter uses the smallest size via that spans all layers and is selected for routing.

center_center

Controls the minimum distance (<*positive_dimension*>) permitted between the centers of any two test points.

If the **center_center** rule is different for two test points, the larger value is used.

If no value is given, center-to-center test point checking is not done.

comp_edge_center

Controls the minimum distance (<*positive_dimension*>) permitted between any test point center and a component boundary edge.

If no value is given, center-to-component edge checking is not done.

image_outline_clearance

Controls the minimum distance (<*positive_dimension*>) permitted between any test point edge and a component boundary edge.

The default is the area-to-testpoint object-to-object clearance specified in the clearance rule.

allow_antenna

Controls whether antennas (stubs) are permitted when test points are added. Antennas are allowed when this rule is **on**.

The default is **on**.

pin_allow

Controls whether through-pins can be used as test points.

When on, you can use **comp** (<*component_id*>) to identify a list of components with through-pins that can be used as test points. If a component list is not included, all through-pins that meet grid and clearance requirements are used.

The default is **off**.

max_len

Restricts the routed length of testpoint antennas. The length is measured from a pad’s origin to the center of the testpoint via.

You can use the **testpoint** rule to improve PCB testability by adding test points to routed signal nets. You can assign the **testpoint** rule by net, class, or for the entire design (pcb). After you set the rule and during the next **route**, **clean**, or **filter** pass, SPECCTRA attempts to mark or add a test point to each net identified in the testpoint rules. For example, a **testpoint** rule at the pcb level can contain settings, and then class or net rules can be used to override these settings.

A test point is a through-pin (pin) or via that SPECCTRA marks as a test point because a **testpoint** rule is set for the net that contains the pin or via. A test via can be a plated-through type or a single surface pad. When an exposed via (not covered by a component body), is not available, SPECCTRA pushes the existing via to an available test point grid site. If this fails, SPECCTRA adds an additional test point via.

Use the **testpoint** rule keywords in conjunction with **insert on** to set the following controls:

- Specify the grid used for placing test vias. The default is the current PCB via grid.
- Identify the probing layer **side** for test points as **front**, **back**, or **both**. You can specify separate **testpoint** rules for the front or back sides of the PCB.
- Specify one or more via padstacks to be used as test points. If you do not use this control, the autorouter chooses a via. Single layer padstacks can be used as test vias.

- Control the minimum center-to-center distance between test points.
- Control the minimum distance between the center of the test point and the component edge (boundary).
- Control the minimum distance between the edge of the test point and the component edge (boundary). You can specify only one **image_outline_clearance** value.
- Control whether antennas (stubs) are allowed.
- Control whether through-pins are used as test points. You can identify a list of components with through-pins that can be used as test points. If a component list is not included, all through-pins that meet the grid and center-to-center requirements are used.
- Control the maximum length of a connection between a net and an inserted test point via.

When you set the minimum test point grid, you can specify a uniform grid or nonuniform X and Y grids. You can specify offsets.

If you change the **testpoint** rule, and run additional **route**, **clean**, or **filter** passes, all test points are redefined based on the current rules. For example, if net sig1 is assigned a test point on the back side and then the **testpoint** rule is changed to front side, SPECCTRA removes the back side test point and attempts to find a test point on the front side after the next **route** or **clean** pass. SPECCTRA does not unmark existing test points for nets where the **testpoint** rule is set to **insert off**.

Using the testpoint rule

Usually, you add test points when your design is routed 100 percent or nearly 100 percent. At this stage, the **testpoint** operation takes advantage of existing vias. A methodology for using **testpoint** with the **route** command in a do file is

- Set all design rules except test point rules
- Run 25 or 50 route passes
- Set test point rules
- Run clean passes
- Run more route passes if necessary
- Run clean passes

For more information about using do files, choose General Information in the Help menu.

Beginning with the next **route**, **clean**, or **filter** pass, a test point is added on a net according to the settings you specify in the **testpoint** rule. For example

- To add a test point on all nets, using a 100 mil grid, enter

  ```
  unit mil
  rule pcb (testpoint (insert on) (grid 100))
  ```

- To add a test point on all nets, using a 100 mil grid that is offset by 25 mil, enter

  ```
  unit mil
  rule pcb (testpoint (insert on) (grid 100 (offset 25)))
  ```

- To add a test point on each net except nets sig1, sig2, and sig3, using a 100 mil grid, enter

```
unit mil
rule pcb (testpoint (insert on) (grid 100))
define (class c1 sig1 sig2 sig3)
rule class c1 (testpoint (insert off))
```

- To add a test point for nets sig1, sig2, and sig3, probed from the back side, using a 100 mil grid; for nets sig4, sig5, and sig6, probed from the front side, using a 200 mil grid; for net sig7, using the current pcb via grid and probing from either side; and for net sig8, using a 75 mil via grid and using padstack TP1, enter

```
unit mil
rule pcb (testpoint (insert off))
define (class back sig1 sig2 sig3)
rule class back (testpoint (insert on) (side back) (grid 100))
define (class front sig4 sig5 sig6)
rule class front (testpoint (insert on) (side front) (grid 200))
rule net sig7 (testpoint (insert on) (side both))
rule net sig8 (testpoint (insert on) (grid 75) (use_via TP1))
```

Notes

The **testpoint** command overrides the pcb **testpoint_rule**. For example, if you enter **testpoint** without options, the operation proceeds with the **testpoint** command default settings, and ignores any rules set at the pcb level with the **testpoint_rule**. Rules set at the higher levels are not affected. To insert test points as a post-processing operation, use the testpoint command.

The clearance rule controls object-to-object clearances for test points, which are edge-to-edge clearances. Special clearances, such as **center_center** and **comp_edge_center** are part of the **testpoint** rule itself and are test point center checks. Test point center checking is a separate checker pass.

The smart_route command does not activate test point insertion until routing is 80 percent complete. You specify the appropriate **testpoint** rule settings and then run **smart_route** with the **auto_testpoint** option.

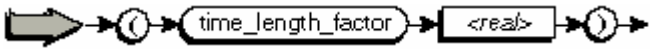
The report testpoint command generates test point summary information. The test point report includes a list of nets that have no **testpoint** rule in effect and those that do have a **testpoint** rule for which SPECCTRA cannot find a test via site. Since the test point feature is disabled for differential pairs, you can also see a list of missing test points for differential pairs in this report.

You can add testpoints to specific nets and wires by using the select net command.

See also the delete command to delete all the test points in a design, including any dangling wiring left by the deletion of a via.

<time_length_factor_descriptor>

The <*time_length_factor_descriptor*> sets a rule that defines the time conversion factor for wire lengths.



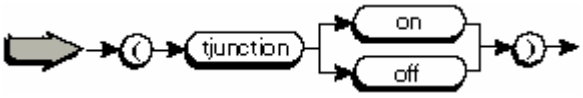
Use **time_length_factor** to set a time conversion factor (<*real*>) for wire lengths. This factor is a ratio of time per unit length and is used as a multiplier to calculate effective wire lengths from delay times.

The conversion factor value must be based on the current measurement units, such as inch or mil and must be consistent with the time units you are using in the design.

You must set a time conversion factor in order for SPECCTRA to follow timing delay rules. See the circuit command for information about setting timing delay rules.

<*tjunction_descriptor*>

The <*tjunction_descriptor*> sets a rule that controls whether tjunctions are permitted in starburst routing.

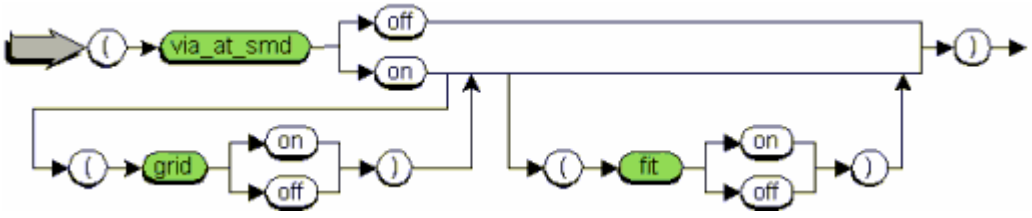


Use the **tjunction** rule to control whether wire tjunctions are permitted in starburst ordered nets. You can allow tjunctions (**on**) or prohibit them (**off**).

When this rule is **on**, you can use junction_type to control whether tjunctions can occur on wire segments, or only on pins, pads, and vias, or only on pins and pads connected to power nets.

<*via_at_smd_descriptor*>

The <*via_at_smd_descriptor*> sets a rule that controls whether escape vias are added under SMD pads.



via_at_smd

Controls whether escape vias are permitted under SMD pads. The choices are

off, which resets a rule to the unspecified state.

on, which permits vias inserted under SMD pads during autorouting.

grid

Controls whether vias inserted under SMD pads are permitted at the pad origin (**off**) or at the via grid point that is nearest the pad origin (**on**).

The default is **off**.

fit

Controls whether vias must completely fit within the SMD pad boundary in order to be inserted under the pad (**on**).

The default is **off**.

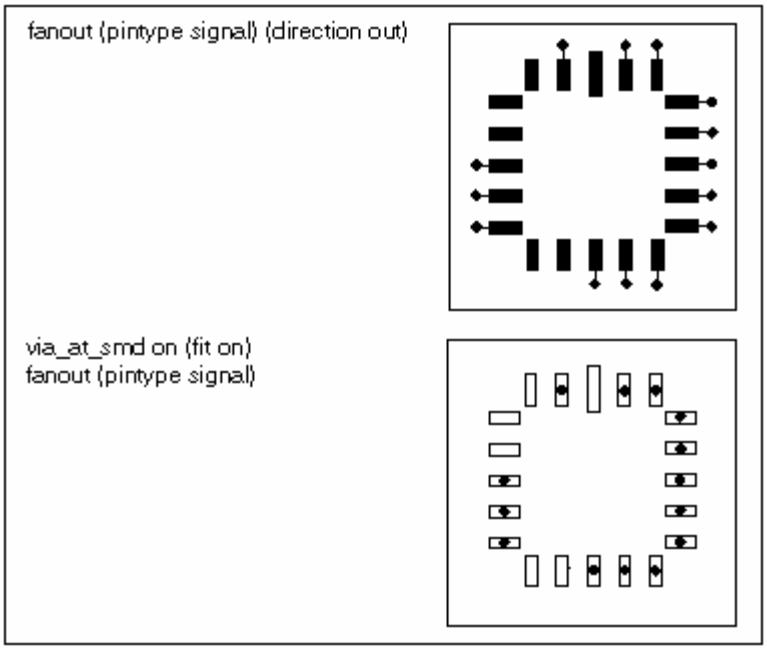
Use the **via_at_smd** rule to control whether escape vias are permitted under SMD pads. You can specify

- Whether escape vias are inserted at the pad origin or at the grid point nearest the pad origin
- Whether escape vias must fit within the pad boundary

If vias are permitted under SMD pads, use a **via_at_smd** rule before using the fanout command. For example, rather than **fanout (pin_type signal) (direction out)**, use the commands

```
rule pcb (via_at_smd on (grid on) (fit on))
fanout (pin_type signal)
```

The different results of **fanout** with and without a **via_at_smd** rule are shown in the following figure.



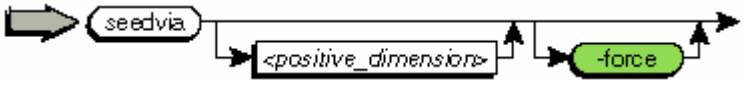
<width_descriptor>

The <*width_descriptor*> sets a rule that controls wire width.

Use the **width** rule to control the width (<*positive_dimension*>) of wires.

seedvia

The **seedvia** command breaks a single diagonal connection into two shorter connections by adding a via.



-force

Adds vias under SMD components on designs with two signal layers.

This command controls the maximum length permitted for diagonal wires. SPECCTRA breaks up two-pin connections that are longer in both X and Y directions than the <*positive_dimension*> you specify.

The seedvia operation adds a single via at a corner of the bounding rectangle for each connection that satisfies the length criteria. At least one through-via that extends through all signal layers must be defined in your design in order to use the **seedvia** command.

Use **-force** if you want the seedvia operation to add vias under SMD components when you route a design with two signal layers.

The default <*positive_dimension*> is 1.0 inch.

Tip

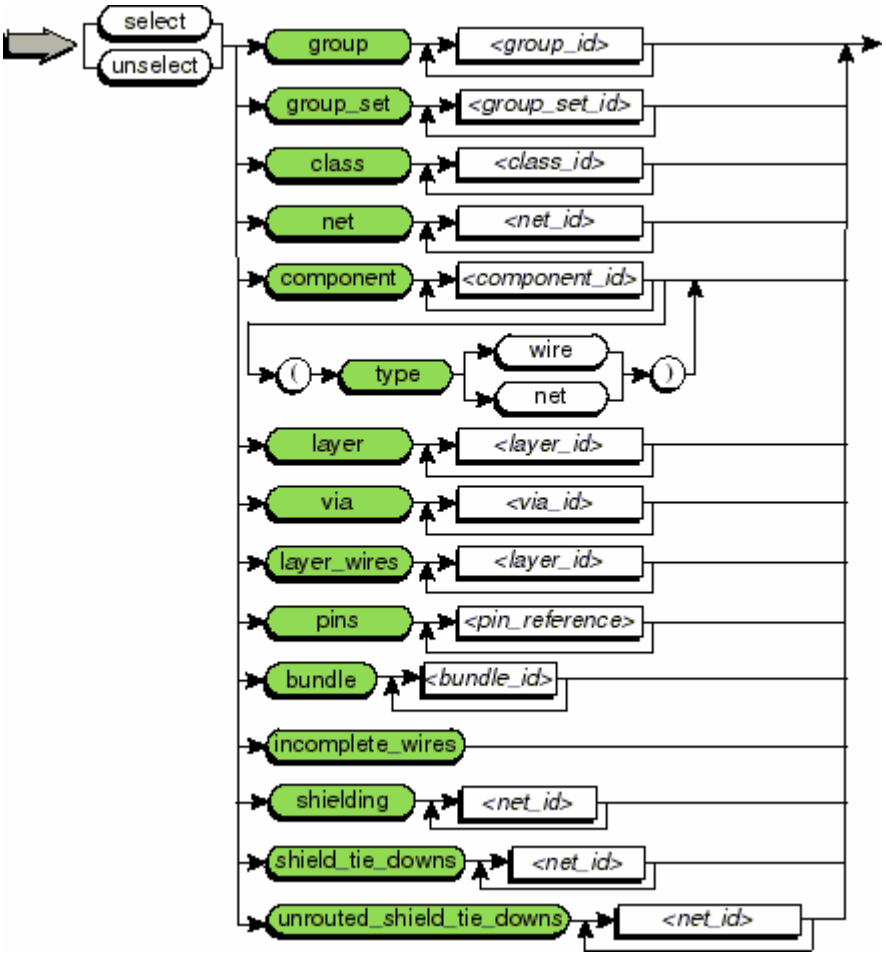
Usually, the **seedvia** command is used for large multilayer designs that are expected to require vias for longer diagonal connections. Because the number of vias can increase dramatically, dependent on the positive dimension you set, a dimension of two inches or more is suggested.

Command examples

seedvia 2
seedvia 2.5 -force

select/unselect

The **select** and **unselect** commands control which connections, vias, and layers are available for autorouting operations.



group

Selects or unselects groups of fromtos. A group consists of one or more fromtos, which are pin-to-pin connections.

group_set

Selects or unselects the groups that belong to group sets.

class

Selects or unselects classes of nets. All pins, vias, wires, and guides in the net are selected or unselected.

net

Selects or unselects nets. All pins, vias, wires, and guides in the net are selected or unselected. Specify the net name (<*net_id*>) exactly as used in the design (same spelling and case).

component

Selects or unselects components. SPECCTRA displays their reference designators.

A reference designator is the reference name assigned to a component in the placement section of the design file.

You can use the **type** option to control whether wires or nets attached to the components are also selected or unselected.

type

Controls which objects attached to the components are selected or unselected. The choices are:

wire, which selects wires attached to pins of the selected or unselected components.

net, which selects nets attached to pins of the selected or unselected components. The pins of other components that share the nets, and the vias that interconnect them, are also selected or unselected.

The default is **wire**.

layer

Selects or unselects one or more layers to control whether the autorouter routes on a specific layer. Selection does not affect layer visibility. The layer name (<*layer_id*>) accepts the question mark (?) and asterisk (*) wildcard characters.

via

Selects or unselects vias, determining which vias can and cannot be available for routing.

Selected vias are available for autorouting. If a via is unselected, it cannot be used unless assigned to a net by a use_via rule in the **circuit** command.

layer_wires

Selects or unselects all wires on specific layers. Only routed wires on these layers are selected or unselected. Guides and component pins are not selected or unselected.

pins

Selects or unselects pins, identifying individual component pins that receive fanout wiring when fanout is initiated. The pin at the other end of the connection is not fanned out, unless you also select it.

bundle

Selects or unselects net bundles created with the **define bundle** command.

incomplete_wires

Incomplete wiring in this sense includes:

pin-to-pin connections with a segment missing. Here, “missing” might or might not include guide wires connecting the other segments.

segments that tee into a pin-to-pin connection but end without completing the connection or end at a guide wire.

segments that start at a pin and end without completing the connection (but segments that end at vias are presumed to be fanouts or test points and are *not* deleted).

wires left dangling by the execution of a **delete conflicts -segment** command.

shielding

Selects or unselects all shield wires and shield tie downs (stub wires that connect shield wires to the shield net) on the specified shielded net (<*net_id*>).

shield_tie_downs

Selects or unselects all routed and unrouted shield tie downs (stub wires that connect shield wires to the shield net) on the specified shielded net (<*net_id*>).

unrouted_shield_tie_downs

Selects or unselects all unrouted shield tie downs (stub wires that connect shield wires to the shield net) on the specified shielded net (<*net_id*>).

Use these commands to select or unselect routing objects for automatic routing. You can

- Select or unselect objects to control which connections are routed during an autorouting operation.
- Select or unselect vias to control whether they are available for a particular autorouting operation.
- Select or unselect pins to control whether they are available for fanout or swapping.
- Select or unselect layers to control whether they are available for a particular autorouting operation.

If you select a layer, the autorouter can use it for routing. If you unselect a layer, the autorouter cannot use it for routing unless a net or class of nets are assigned to the unselected layer with a use_layer rule. Nets that are assigned a routing layer with the use_layer rule are always routed on the assigned layer whether the layer is selected or unselected.

If there are SMDs in the design, and these components are mounted on an unselected layer (front or back), the autorouter routes short escape wires and vias on the unselected layer. See **smd_escape** in the change command for information about setting the length of the escape wires.

At the beginning of a SPECCTRA session, all objects except layers and vias are unselected by default. Initial layer and via selection status for autorouting is based on <*layer_descriptor*> and <*via_descriptor*> entries in the design file.

You can select nets, classes, groups, group sets, or components for use in certain automatic and interactive routing operations. When nets or fromtos are selected, only

these connections are available for autorouting operations. For instance, if you select one or more nets and use the **route** command, only these nets are routed. Other (unselected) objects are not affected. If no nets or fromtos are selected, which means all objects are unselected, then all objects are available for autorouting operations.

SPECCTRA displays selected objects in the select color, which is yellow if you are using the default color map.

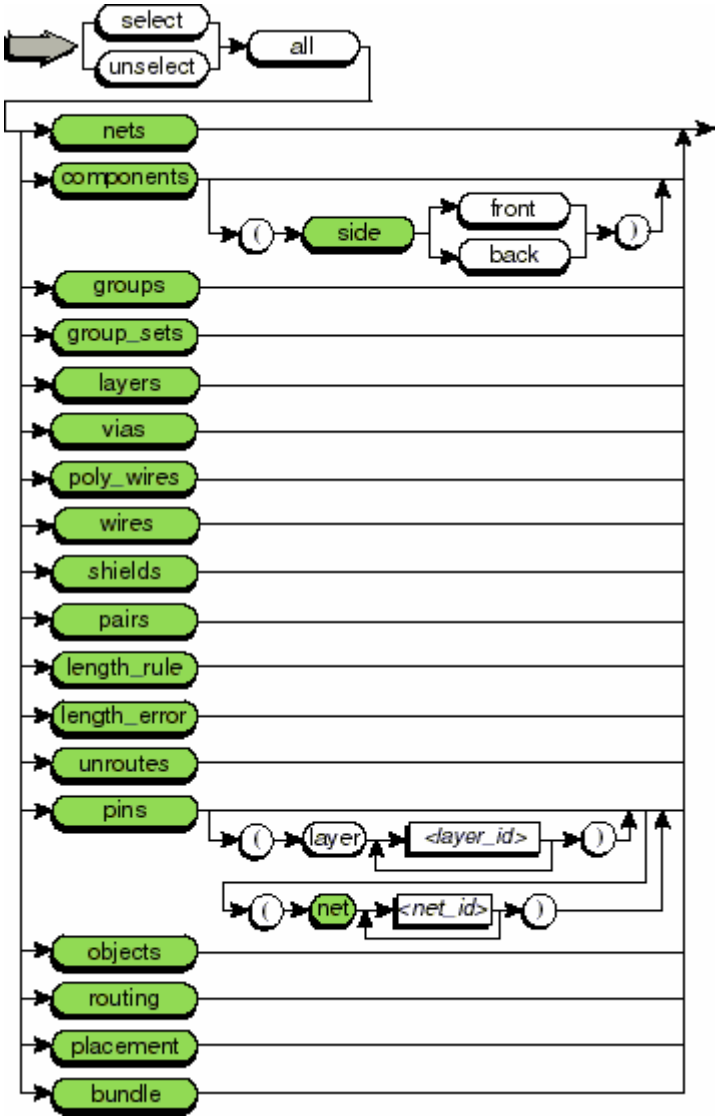
Command examples

```
select group G1
select net ABUS??
select class CLKS1 CLKS2
select component U2 (type net)
select layer L1 L4
select via V27
select group_set grpset1
select layer_wires S1 S2
select pins U1-5 U3-6

unselect layer L5 L6
unselect via V50
```

select/unselect all

The **select all** and **unselect all** commands control whether connections attached to all objects of a particular type, or all vias or layers are available for autorouting operations.



nets

Selects or unselects all nets in the design. All guides, vias, wires, and pins with nets attached are selected or unselected.

components

Selects or unselects all components on one or both sides of the PCB. You can specify **front** or **back**.

side

Controls whether the current operation applies to the front side (**front**) or the back side (**back**) of the PCB. By default, the operation applies to both sides of the PCB.

groups

Selects or unselects all groups of fromtos. A group consists of one or more fromtos,

which are pin-to-pin connections.

group_sets

Selects or unselects groups that belong to all group sets.

layers

Selects or unselects all signal layers defined in the design file. Selecting layers makes them available for routing and other operations.

vias

Selects or unselects all vias defined in the design file. Selecting a via makes it available for use during autorouting.

poly_wires

Selects or unselects only wiring polygons. Other wire objects are not affected by this option.

wires

Selects or unselects all wires, including wiring polygons. All pins, guides, and vias connected to the wires are also selected or unselected.

shields

Selects or unselects all nets that have assigned shields. All guides, vias, wires, and pins attached to the nets are also selected.

pairs

Selects or unselects defined differential pairs. All pins, vias, wires, and guides in both nets of the differential pairs are selected or unselected.

length_rule

Selects or unselects all nets assigned length rules, which includes minimum and maximum length rules and matched length rules. For a net length rule, the entire net is selected. For a fromto length rule, only the fromto is selected.

length_error

Selects or unselects all nets with length rule violations, which includes minimum and maximum length rule violations and matched length rule violations. For a net length rule violation, the entire net is selected. For a fromto length rule violation, only the fromto is selected.

unroutes

Selects or unselects guides for all unrouted connections.

pins

Selects or unselects all component pins in the design, on certain layers, or connected

to certain nets on one or more layers.

You can select all pins on certain layers by using the **layer** keyword and specifying one or more layer names (<*layer_id*>). You can select all pins connected to certain nets on a layer by using the **net** keyword and specifying one or more net names (<*net_id*>).

The default is all component pins in the design.

Note

Use this option to specify the component pins you want to receive fanout wiring when you run the **fanout** command.

net

Selects or unselects component pins connected to one or more nets (<*net_id*>).

objects

Selects or unselects all routing and placement objects.

routing

Selects or unselects all routing objects. All components, pins, guides, and vias connected to the wires are also selected or unselected.

placement

Selects or unselects all placement objects.

bundle

Selects or unselects all net bundles defined in the design file or with the **define bundle** command.

Use these commands to select or unselect all objects of a certain type for autorouting. You can

- Select or unselect all objects of a particular type to control whether connections attached to those objects are routed during a particular autorouting operation.
- Select or unselect all objects of a particular type and protect them so they cannot be deleted, ripped up, or rerouted.
- Select or unselect all vias to control whether they are available for a particular autorouting operation.
- Select or unselect all pins to control whether they are available for fanout and swapping.
- Select or unselect all layers to control whether they are available for a particular autorouting operation.

To select or unselect all component pins for fanout on certain layers only, identify one or more layer names.

At the beginning of a SPECCTRA session, all objects are unselected by default. Layer and via availability for autorouting depends on <*layer_descriptor*> and <*via_descriptor*> entries in the design file.

You can select objects for certain automatic and interactive routing operations. When objects are selected, only these objects are available for autorouting operations. Other (unselected) objects are not affected. If no objects are selected, which means all objects are unselected, then all objects are available for autorouting operations.

SPECCTRA displays selected objects in the select color, which is yellow if you use the default color map.

Notes

You do not need to issue **select all nets** before you begin autorouting. If nothing is selected (default when the design is loaded), all nets are processed by any autorouter operation. If one or more nets are selected, the autorouter processes only the selected nets.

Initially, all signal layers (except any layers unselected in the design file) are enabled for routing and other operations when you start SPECCTRA. You do not need to issue **select all layers** unless you want to reverse a prior **unselect layer** command.

On some layout systems, not all of the vias defined in a design are available for autorouting. By default, in the SPECCTRA design file, only those vias that are available for routing in the layout system are selected. Vias identified as spares in a design file <*via_descriptor*> are not selected. You can override the design file defaults by selecting all vias in the design.

If you want to use a particular via that is not the default used by the autorouter, you can use the commands

```
unselect all vias
select via <via_id>
```

The <*via_id*> is the padstack name for the via you want to use.

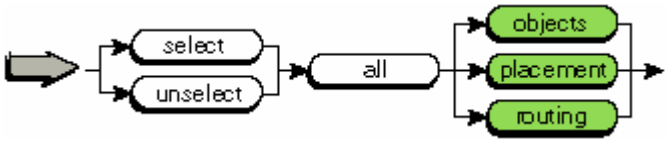
Command examples

```
select all wires
select all poly_wires
select all components (side front)
select all groups
select all group_sets
select all shields
select all pairs
select all length_rule
select all unroutes
select all pins (layer s1 s2)

unselect all nets
unselect all vias
unselect all layers
```

select/unselect all objects

The select all objects and unselect all objects commands select or unselect all routing objects, placement objects, or both.



objects

Selects or unselects all routing and placement objects.

placement

Selects or unselects all placement objects.

routing

Selects or unselects all routing objects. All components, pins, guides, and vias connected to the wires are also selected or unselected.

Use these command to select all unselected objects or to unselect all selected objects. You can select or unselect all placement objects, all routing objects, or both

When you select placement objects, only the selected objects are available for placement operations. When you select routing objects, only those objects are available for routing operations.

For general information about using select and unselect commands, see selecting placement objects.

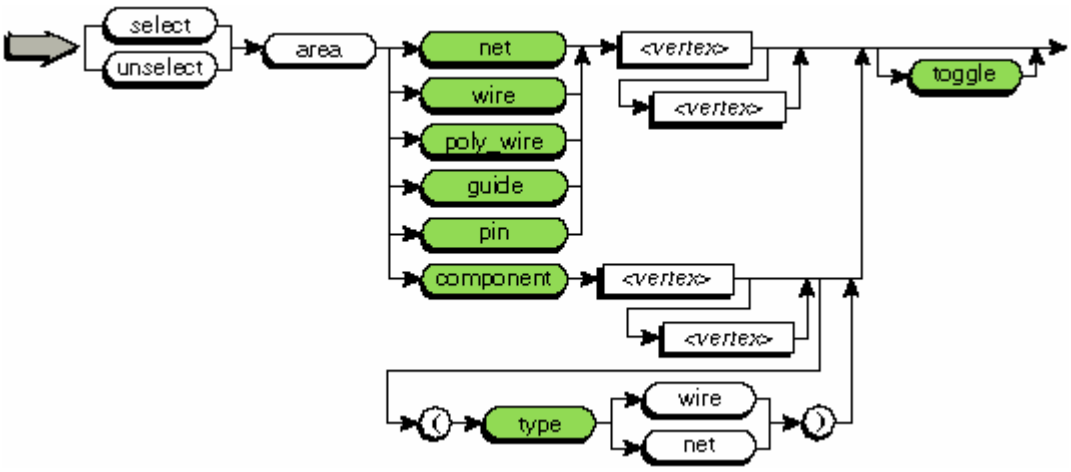
Command examples

select all placement

unselect all objects

select/unselect area

The select area and unselect area commands select or unselect objects at a specific location or area.



net

Selects or unselects nets that are totally or partially within the defined area. All pins, wires, and guides attached to the selected nets are also selected or unselected.

wire

Selects or unselects wires that are totally or partially within the defined area. All pins, vias, and guides attached to the selected wires are also selected or unselected.

poly_wire

Selects or unselects wiring polygons that are totally or partially within the defined area. All pins, vias, and guides attached to the selected wiring polygons are not selected or unselected.

guide

Selects or unselects guides within the defined area. Guides are pin-to-pin connections that are not routed.

pin

Selects or unselects all component pins within the defined area, specifying that these pins receive fanout wiring when fanout is initiated.

component

Selects or unselects components within the defined area. SPECCTRA displays their reference designators. A reference designator is the reference name assigned to a component in the placement section of the design file.

You can use the **type** option to control whether wires or nets attached to the components are also selected or unselected.

type

Controls which objects attached to the components are selected or unselected. The choices are:

wire, which selects wires attached to pins of the selected or unselected components.

net, which selects nets attached to pins of the selected or unselected components. The pins of other components that share the nets, and the vias that interconnect them, are also selected or unselected.

The default is **wire**.

toggle

Switches the selection state of the objects you are selecting within the defined area. All currently selected objects become unselected, and all currently unselected objects become selected. Does not affect any objects other than the type you are selecting.

This option is valid with the **select area** commands but not with the **unselect area** commands.

Use these commands to select or unselect objects for autorouting operations. You can

- Select or unselect an object at a specific location.
- Select or unselect all objects of a particular type within an area.

Use <*vertex*> to identify the X and Y coordinates of a location or area where you want to select or unselect objects.

- Specify the coordinates for a point within the bounds of an object that you want to select or unselect.
- Specify the coordinates for two diagonally opposed corners of a rectangular area to select or unselect the objects within its boundary.

Use the **toggle** option to switch the selection state of objects within an area. This option can be used with the **select area** command but not with the **unselect area** command.

When you select components in an area, you can control whether all wires or all nets attached to these components are available for autorouting operations.

When you select pins, you can control whether all pins within the area are available for fanout.

At the beginning of a SPECCTRA session, all objects are unselected by default.

You can select objects for certain automatic and interactive routing operations. When objects are selected, only these objects are available for autorouting operations. Other (unselected) objects are not affected. If no objects are selected, which means all objects are unselected, then all objects are available for autorouting operations.

SPECCTRA displays selected objects in the select color, which is yellow if you are using the default color map.

Command examples

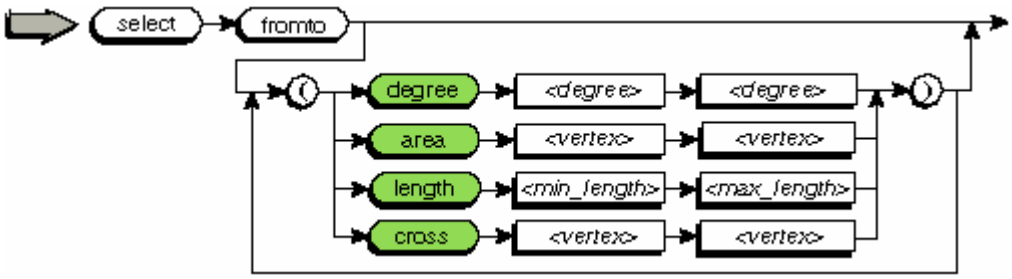
select area net 1.325 4.350

select area net 8.855 5.440 11.75 2.63 toggle
select area component 8.345 5.550
select area component 0.600 0.225 1.025 0.600 (type net)
select area pin 2.25 3.50 3.75 4.25

unselect area wire 8.855 5.440 11.75 2.63
unselect area guide 3.35 .650 1.375 1.9

select/unselect fromto

The select fromto and unselect fromto commands control which fromtos are available for autorouting operations.



degree

Selects or unselects fromtos located within the specified range of angles. Angles are measured counterclockwise. The positive dimension must be from 0 to 360 degrees.

area

Selects or unselects the fromto for each pin located within the area defined by two vertexes.

length

Selects or unselects a fromto if its diagonal length falls between the specified minimum and maximum length limits.

cross

Selects or unselects fromtos that cross the area defined by two vertexes.

Use these commands to select or unselect all routed fromtos, or fromtos that meet certain requirements.

Use <*degree*> to identify a range of angles. You can select or unselect fromtos located within this range.

Use <*vertex*> to specify the coordinates for two diagonally opposed corners of a rectangular area. You can select or unselect fromtos located in this area, or crossing this area.

Use <*min_length*> and <*max_length*> to specify minimum and maximum length limits. You can select or unselect fromtos with a diagonal length within this range.

At the beginning of a SPECCTRA session, all objects are unselected by default.

You can select objects for certain automatic and interactive routing operations. When objects are selected, only these objects are available for autorouting operations. Other (unselected) objects are not affected. If no objects are selected, which means all objects are unselected, then all objects are available for autorouting operations.

SPECCTRA displays selected objects in the select color, which is yellow if you are using the default color map.

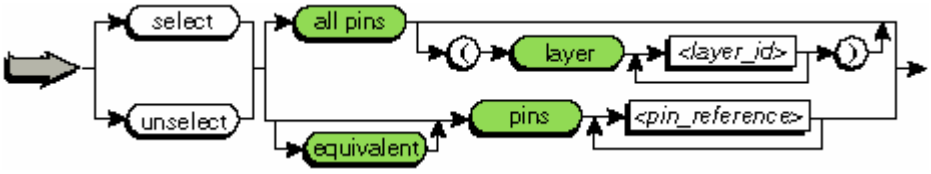
Command examples

select fromto
select fromto (degree 80 100)
select fromto (area 30 65 170 -25) (degree 170 190)
select fromto (length 75 125)
select fromto (cross -100 -72 -20 -72)

unselect fromto
unselect fromto (degree 80 100)

select/unselect pin

The **select pin** and **unselect pin** commands select and unselect component pins for subsequent placement or routing operations.



all pins

Selects or unselects all component pins in the design (by default) or on one or more specified layers (if you use the **layer** option).

Wires and guides attached to the selected pins are also selected.

layer

Selects or unselects all component pins on one or more specified signal or power layers (<*layer_id*>).

Wires and guides attached to the selected pins are also selected.

pins

Selects or unselects one or more component pins. Wires and guides attached to the selected pins are also selected.

equivalent

Selects or unselects the specified pins, and their equivalents anywhere in the design.

Wires and guides attached to the selected pins are also selected.

Pins are equivalent if they perform the same logical function, whether they have the same or different names.

These commands select or unselect component pins.

- When you use the **all** option, you select or unselect all the pins in the design. You can use the **layer** option to select all the pins on one or more layers. A <*layer_id*> is the name of a signal or power layer defined in the design file.
- When you use the **pins** option, you select or unselect only the pins you specify by pin reference. The <*pin_reference*> consists of the component reference designator and the pin number of a pin that belongs to the component. You can use the **equivalent** option to select both the pins you specify and their equivalents anywhere in the design.

When you select pins, only the net connections to the selected pins are available for swap or fanout operations. For general information about using select and unselect commands, see selecting placement objects.

See also

swap
fanout
select area commands.

See the <*part_library_descriptor*> in the *SPECCTRA Design Language Reference* for information about gate and pin swapping.

Command examples

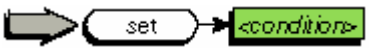
select pins U1-5 U1-6 U2-4 U2-5

select all pins

select equivalent pins U2-8 U6-3 U23-4

set

The **set** command controls how layers display, the status file update interval, and several autorouting options.



There are a variety of conditions you can control by using the **set** command. You can set how layers display, the status file update interval, and autorouting options, including crosstalk model and crosstalk report options.

Some settings require on or off switches, and some require values or option choices. Use <*condition*> to set your conditions. See set conditions overview for general information about controlling conditions with the **set** command.

Notes

SPECCTRA updates the status file after every 100 connections are routed. If you reduce the interval, routing time increases.

When SPECCTRA loads your design file, all layers are superimposed, and displayed as one composite image. Alternatives are

- You can display between 1 and 8 layers panels. If you specify more than 8 or if you enter a value less than 1, **graphing** defaults to 1.
- Only layers currently selected for viewing are included in the display.
- If you have more than eight layers selected for viewing, the additional layers are superimposed on the first eight selected layers.
- When **graphing** is set to 3, 5, or 7, the screen is partitioned into 4, 6, or 8 areas respectively, although only the number of panels specified are displayed.
- Zoom does not work when graphing is set greater than 1.

When performing same net violation checking, if an ambiguous situation occurs, SPECCTRA might flag a net as having a violation when no violation exists. Visually review the violations to make this determination.

The soft fence setting is useful for separating analog and digital signals. When setting hard and soft fences, remember that hard and soft fences cannot coexist. Either all fences are hard or all are soft. See also the fence command.

By default, pin width tapering occurs as a post-processing step. To perform pin width tapering during autorouting, enter **set search_tapering on** before using routing commands. For more information about pin width tapering, see the pin_width_taper rule.

For more information about density analysis, see the density analysis command.

By default, SPECCTRA considers one adjacent signal layer for tandem noise and segment crosstalk calculations. You can use **tandem_depth** to control how many adjacent signal layers are considered, but usually **tandem_depth** is not set higher than 2. A <*depth*> value higher than 1 slows the router.

Command examples

```
set update_interval 200
set gather_wires off
set graphing 4
set noise_accumulation RSS
set soft_fence on
set same_net_checking on
set search_tapering on
set force_to_terminal_point on
set routability_colors 15
set default_net_coupling friendly
set diagonal_mode on
```

set tandem_depth 2
set dynamic_zoom off
set min_selection on

<condition>

auto_merge_polygon

Use **set auto_merge_polygon** [**on** | **off**] to control automatic polygon merging for interactive routing.

This condition enables automatic merging of polygons with same properties that are overlapped during interactive move operations. Wiring polygons are merged only if they belong to the same net and occupy the same layers. Keepouts are merged only if they are the same type and on the same layers, and have the same rules. The choices are

off, which means the polygons are not merged (default)

on, which means the polygons are merged if the polygon merge conditions are satisfied

For example

set auto_merge_polygon on

average_pair_lengths

The **set average_pair_lengths** option controls how the routed lengths of paired nets are calculated for rule checking.

This condition controls whether SPECCTRA considers the average routed length for the pair when checking timing rules (length and delay). The average length is calculated by adding the individual lengths of the two wires in the pair, and dividing by two. The choices are

on, which means check the average length of the paired nets for timing rule violations.

off, which means check each net independently for timing rule violations.

The default is **on**.

For example

set average_pair_lengths off

bbv_ctr2ctr

Use **set bbv_ctr2ctr** [**on** | **off**] to control how SPECCTRA measures gap and stagger distances for blind and buried vias.

This condition controls whether SPECCTRA measures gap and stagger distances between vias from the via centers (**on**) or the via edges (**off**). The default is **off**.

For example

set bbv_ctr2ctr on

Note

Rules affected by this condition include **clearance** rules with the **type buried_via_gap** option. See the rule command for details.

crosstalk_model

Use **set crosstalk_model** <*model_name*> to choose the crosstalk model for routing and rule checking.

This condition controls which crosstalk model is used in crosstalk routing rules. The <*model_name*> choices are

cct1, which uses parallel and tandem noise rules to control routing and report cumulative noise violations on routed nets.

cct1a, which uses parallel and tandem noise rules with noise saturation factors (see rule <saturation_length_descriptor> for details) to control routing and report cumulative noise violations on routed nets.

The default is **cct1**.

For example

```
set crosstalk_model cct1a
```

See also

diagonal_mode

Use **set diagonal_mode** [**on** | **always** | **off**] to control diagonal routing.

This condition controls whether the autorouter uses diagonal routing. The choices are

on, which means the autorouter can route diagonally when it needs to during diagonal memory routing, through staggered pin arrays, and near existing diagonal wires. In general, this option does not produce much diagonal routing.

always, which means the autorouter routes every wire with long diagonals, depending on the amount of available routing space. This option causes the majority of wires to have diagonals.

off, which means the autorouter never uses diagonal routing and routes only orthogonal wires.

The default setting is **on**.

Examples:

```
set diagonal_mode always
```

Note

Diagonal routing is generally slower than orthogonal routing. Performance is degraded particularly if you use **set diagonal_mode always**. Tough, dense designs will probably not benefit from this option.

dofile_auto_repaint

Use **set dofile_auto_repaint** [**on** | **off**] controls repaints when you run commands from a do file.

This condition controls whether SPECCTRA repaints the work area after operations performed by commands in a do file. The choices are

on, which means repaint the work area after operations performed by a do file.

off, which means do not repaint the work area after operations performed by a do file.

The default is **on**.

For example

```
set dofile_auto_repaint on
```

You can use this control by including it in a do file, entering it in the command entry area, or clicking **View - Dofile Repaints**.

Tip

Turning off **dofile_auto_repaint** affects only operations performed by commands in do files. If you have turned off this control but you want a do file to repaint the work area after running a certain command, include the **repaint** command in the do file. For example

```
clean 2
repaint
```

Notes

You can use the **set repaint** option to control whether SPECCTRA repaints the work area after operations you perform with the mouse, by choosing commands from a menu, or by entering commands in the command entry area.

dynamic_pinswap

Use **set dynamic_pinswap** [**on** | **off**] to control whether SPECCTRA can swap net pin connections during autorouting operations.

This condition controls whether the autorouter attempts to swap net connections on pins during **route** and **clean** passes. The choices are

on, which permits pin swapping during autorouting operations.

off, which prohibits pin swapping during autorouting operations.

The default is **off**.

For example

```
set dynamic_pinswap on
```

Pin swapping is useful for designs with DIE components, BGA components, or plating bars. After each pass, the autorouter looks for crossed wires, and attempts to uncross them by swapping the net pin connections and rerouting the wires.

Notes

The necessary package swap information must be translated from your layout system and included with the component definitions in the SPECCTRA design file.

The autorouter cannot swap pin connections on components after you have changed their images, added net connections to pins, or removed net connections from pins. See the change component_image and define net pins commands for details.

Connections on locked pins are not swapped. See the lock command for details.

dynamic_zoom

Use **set dynamic_zoom** [**on** | **off**] to control pan and zoom operations in the work area.

This condition controls dynamic pan and zoom of the display. The choices are

off, which prevents dynamic panning and zooming. Static pan and zoom remains available.

on, which enables dynamic panning and zooming.

The default depends on a setting in the specctra.ini file, or on the determined speed of the workstation.

If there is a specctra.ini file with the [GUI] section setting "AllowDynamicZoom=1", the initial state is **on.** If "AllowDynamicZoom=0", the initial state is **off**. If there is no AllowDynamicZoom setting in the specctra.ini file, then the default dynamic zoom setting depends on the estimated speed of the host PC.

edit_abort_uses_undo

Use **set edit_abort_uses_undo** [**on** | **off**] to control whether the interactive router can undo pushed wires or vias when you cancel a wire edit.

This condition controls whether the interactive router can undo pushed wires and vias in Edit Route mode when you use the **Cancel** command in Edit Route popup mode. The choices are

on means pushed wires and vias are restored to their previous positions before you started the wire edits you are canceling.

off means pushed wires and vias remain where they were pushed during the edits you are canceling.

The default is **off**.

For example

```
set edit_abort_uses_undo on
```

force_to_terminal_point

Use **set force_to_terminal_point** [**on** | **off**] to control how wires are routed on pins.

This condition controls whether the autorouter routes to the origin of a pin (**on**) or to a point on any part of a pin shape (**off**). The default is **off**.

For example

set force_to_terminal_point on

gather_wires

Use **set gather_wires** [**on** | **off**] to control how wires connect to differential pairs or bundles.

This condition controls whether extra wire bends are eliminated when connecting the wires of a differential pair or a bundle to pins. The choices are

off, which means the extra bends can occur.

on, which means the extra bends are removed.

The default is **on**.

For example

set gather_wires on

graphing

Use **set graphing** <*positive_integer*> to control whether your design is displayed in split views by layer or in a single composite image.

This condition controls how your design is displayed in the work area. You must specify the number of layers (<*positive_integer*>) you want to display separately (rather than the default composite image where all layers are superimposed). This value must be a positive integer from 1 to 8. If the value is greater than 8 or less than 1, **graphing** defaults to 1.

For example

set graphing 4

include_pins_in_crosstalk

Use **set include_pins_in_crosstalk** [**on** | **off**] to control whether pins are considered in noise calculations.

This condition specifies whether pin shapes are included in the measurements for calculations that use parallel_noise and tandem_noise rules.

The default is **off**.

For example

set include_pins_in_crosstalk on

microvia

Use **set microvia** [**on** | **off**] to control the availability of licensed MicroVia features.

This condition controls the availability of MicroVia features at the command line and in the Graphical User Interface. MicroVia features made available by this set command require a special license .

Use **set microvia on** when you plan to incorporate microvias in your design. The following features become available:

Enhanced fanout

This feature provides improved fanout for vias under SMD pads when pads may be directly opposite each other on opposite sides of the board. See the note in

fanout command

Stacked vias

This feature allows stacking of blind and buried vias at the same location on different layers, and provides enhanced support for depth control. See

stack_via rule
stack_via_depth rule
stack_via_depth report

Via arrays

This feature provides the capability to define a template for via arrays in the design file which works with a circuit rule to create via arrays automatically during automatic routing. Additional features enable interactive modification of via arrays. See

<*via_array_template_descriptor*> in the *Design Language Reference*
use_via circuit rule
change via mode
rotate via mode

min_selection

Use **set min_selection** [**on** | **off**] to control area selection of cut segments in a wire.

This condition controls the behavior of the **select area wire** command to enable selection of cut segments in a wire. The choices are

on, which limits selection of segments up to the first two-way pseudopin, via, t-junction, or pin in both directions along the wire. This limits selection to a cut segment when the cut segment terminates in a two-way pseudopin.

off, which limits selection of segments up to the first via, t-junction, or pin in both directions along the wire from the selection point. This is the default and normal **select area wire** behavior.

Note

A two-way pseudopin exists where a wire segment is cut (cut segment mode) when the **cut_mode_splits_wires** option to the **set** command is on. This option is available on the Cut Segment Mode [RB] menu.

noise_accumulation

Use **set noise_accumulation** [| **RSS**] to control accumulated noise calculations for a net.

This condition sets the method used for calculating the total noise accumulated on a net. The choices are

linear, which means the accumulated noise on a victim net is just the simple sum of the noise contributions of the individual aggressor nets.

RSS, which means the accumulated noise on a victim net is the square root of the sum of the squares of the noise contributions of the individual aggressor nets.

The default is **linear**.

For example

```
set noise_accumulation RSS
```

noise_calculation

Use **set noise_calculation** [| **linear_interpolation**] to control noise interpolation from a user-specified noise table.

This condition sets the interpolation method for a user-specified noise table. The choices are

stairstep, which means interpolated values are calculated for fixed ranges between supplied values.

linear_interpolation, which means interpolated values are calculated at exact points between supplied values.

The default is **stairstep**.

For example

```
set noise_calculation linear_interpolation
```

repaint

Use **set repaint** [**on** | **off** | **manual**] to control when repaints are performed.

This condition controls when SPECCTRA repaints the work area. The choices are

on, which permits all repaint operations.

off, which prohibits all repaint operations.

manual, which permits repaint operations only when you use the **repaint** command or perform a viewing operation such as zoom or pan.

The default is **on**.

For example

```
set repaint manual
```

Note

You can use the **dofile_auto_repaint** option to control whether SPECCTRA repaints the work area after operations performed by commands in a do file.

reroute_order_viols

Use **set reroute_order_viols** [**on** | **off**] to control whether the autorouter attempts to rip up and reroute nets with order violations.

By default after each routing pass, the autorouter the autorouter attempts to reroute the nets that have order violations. Order violations can occur during autorouting or when you read a routes file or wire file that contains incorrectly ordered wires. Use

set reroute_order_viols off if you want to prevent the autorouter from rerouting the nets. The default is **on**.

For example

```
set reroute_order_viols off
```

Note

The routing status report lists the number of net order violations for each routing pass. Use **report order_violations** to generate a list of order violations. To visually display the order violations, use setup_check to turn on order rule checking, and run the check command. You can also use the highlight command to display the order violations.

When you run **check** with order rule checking turned on, SPECCTRA reports the number of net order rule violations in the output window.

rotate_jumper_via

Use **set rotate_jumper_via** [**on** | **off**] to control whether SPECCTRA can rotate nonsymmetrical jumper vias.

This condition controls whether the autorouter can rotate nonsymmetrical jumper vias 90 degrees if necessary for proper routing of the jumper wires. The choices are

on, which permits jumper via rotation.

off, which prohibits jumper via rotation.

The default is **off**.

For example

```
set jumper_via on
```

roundoff_rotation

Use **set roundoff_rotation** [**on** | **off**] to control how the autorouter calculates pad rotation coordinates.

This condition controls whether the autorouter rounds off pad locations to the nearest coordinate when rotating non-circular pads at angles that are not multiples of 90 degrees. The choices are

on, which means the autorouter rounds off the pad locations to the nearest coordinate.

off, which means the autorouter truncates extra decimal places to calculate the pad locations.

The default is **off**.

For example

```
set roundoff_rotation on
```

routability_colors

Use **set routability_colors** <*positive_integer*> to set the color scale used in the

density analysis display.

This condition controls the number of color gradations (<*positive_integer*>) that are used in the color scale chart for the density analysis display. This value must be between 2 and 20, inclusive.

For example

```
set routability_colors 6
```

See also the density analysis command.

same_net_checking

Use **set same_net_checking** [**on** | **off**] to control rule checking for clearance violations between objects on the same net.

This condition controls whether SPECCTRA checks for clearance rule violations between objects on the same net. A same net clearance rule violation occurs when a wire segment, via, or pin is too close to another object on the same net.

The choices are

on, which enables checking for clearance rule violations between objects on the same net.

off, which disables checking for clearance rule violations between objects on the same net.

The default is **off**.

For example

```
set same_net_checking on
```

Note

The **via_via** and **via_via_same_net** clearance rules are always checked and are not affected by this control.

search_tapering

Use **set search_tapering** [**on** | **off**] to control automatic pin width tapering.

This condition controls whether the autorouter performs pin width tapering during the autorouting phase (**on**) or during the post-processing phase (**off**).

The default is **on**.

For example

```
set search_tapering on
```

shadow_mode

Use **set shadow_mode** [**on** | **off**] to control how selected nets and components are displayed.

This condition controls whether the SPECCTRA distinguishes selected objects by displaying them in a special select color (yellow in the default color map) or by shadowing the colors of unselected objects. The choices are

on, which displays selected objects in the select color and unselected objects in their layer colors

off, which displays selected objects in their layer colors and unselected objects in a dimmed (shadow) representation of their layer colors.

The default is **off**.

For example

set shadow_mode on

show_snap_grid_cursor

Use **set show_snap_grid_cursor** [**on** | **off**] to control whether SPECCTRA displays the snap grid cursor for interactive [LB] modes.

This condition controls the visibility of the snap grid cursor, a small white square that shows the snap grid point nearest to the pointer) when a snap grid has been defined. The snap grid cursor appears in Measure mode, Add/Edit Polygon mode, Move mode, Copy Polygon mode, Cut Segment mode, Cut Polygon mode, and the Draw modes (Fence, Keepout, Region, Place Boundary, Room, and Ruler). When you move the pointer closer to a different snap grid point, the snap grid cursor appears over that grid point. The choices are

on, which displays the snap grid cursor.

off, which hides the snap grid cursor.

The default is **on**.

For example

set show_snap_grid_cursor off

Note

The color of the snap grid cursor is controlled by the highlight color, which is white in the default color map.

See also

grid snap

soft_fence

Use **set soft_fence** [**on** | **off**] to control how fences affect autorouting.

This condition controls whether fences are soft (**on**) or hard (**off**).

A soft fence causes the autorouter to do the following:

Route all connections inside the soft fence within the fence boundary

Route all connections outside the soft fence outside the fence boundary without crossing the fence

Ignore the fence for all connections that cross the soft fence

A hard fence causes the autorouter to route only connections that are completely inside the fence.

The default is **off**.

For example

set soft_fence on

stub_viols_costs

Use **set stub_viols_costs** [**on** | **off** | <*positive_integer*>] to control how many stubs the autorouter can route with maximum stub length rule violations.

This condition controls the cost for routing stubs that are longer than the max_stub rule. The default is **on**, and the permitted number of stub length violations is set internally. You can use <*positive_integer*> to increase this number and improve the autorouter completion rate. Use **off** if you do not want to permit stub length rule violations.

For example

set stub_viols_costs 2

set stub_viols_costs off

Note

The routing status report lists the number of stub length rule violations for each routing pass. Use **report order** to generate a list of stub length rule violations.

To visually display the violations, use setup_check to turn on stub rule checking, and run the check command. You can also use the highlight command to display the violations.

When you run **check** with stub rule checking turned on, SPECCTRA reports the number of stub rule violations in the output window.

swap_fanouts

Use **set swap_fanouts** [**on** | **off**] to control whether SPECCTRA can swap fanout connections on pins during autorouting operations.

This condition controls whether the autorouter attempts to swap net connections on pins with fanouts during **route** and **clean** passes. The choices are

on, which permits pin swapping of fanouts during autorouting operations.

off, which prohibits pin swapping of fanouts during autorouting operations.

The default is **off**.

For example

set swap_fanouts on

The autorouter can attempt to swap fanout connections between pins if each pin is attached to a single, unprotected fanout wire, the wires are not protected and have the same widths and wiring rules, and the fanout vias have the same via padstacks and via rules and are not attached to any other wires.

Notes

Connections on locked pins are not swapped. See the lock command for details.

The necessary package swap information must be translated from your layout system and included with the component definitions in the SPECCTRA design file.

The autorouter cannot swap pin connections on components after you have changed their images, added net connections to pins, or removed net connections from pins. See the change component_image, and define net pins commands for details.

See also set dynamic_pinswap.

tandem_depth

Use **set tandem_depth** <*positive_integer*> to control the layer depth for noise and crosstalk calculations.

This condition controls the number of adjacent signal layers (<*positive_integer*>) considered in tandem noise and segment crosstalk calculations. This setting applies to tandem_noise and tandem_segment rules. The default is 1. A value less than or equal to 0 means the default is used.

For example

```
set tandem_depth 2
```

Note

An adjacent layer separated by a power layer is not considered even if it falls in the layer range controlled by the <*depth*> value.

unknown_user_property_warning

Use **set unknown_user_property** [**on** | **off**] to control whether the unknown property warning is enabled.

This condition controls whether SPECCTRA issues a warning when you define a user property by specifying a property name that SPECCTRA does not recognize (**on**) or does not issue a warning (**off**). The default is **on**.

For example

```
set unknown_user_property_warning off
```

update_interval

Use **set update_interval** <*positive_integer*> to control how often the status file is updated when you run the autorouter.

This condition controls the frequency of updates to the status file. You must specify the number (<*positive_integer*>) of connections to be routed before SPECCTRA updates the status file. By default, SPECCTRA automatically updates the status file after every 100 connections are routed.

For example

```
set update_interval 200
```

via_to_layer_pattern

Use **set via_to_layer_pattern** [**on** | **off**] to control whether SPECCTRA uses the layer fill pattern to display vias on a layer.

This condition controls which fill pattern SPECCTRA uses to display vias on a layer. The choices are

on, which means fill vias with the layer fill pattern.

off, which means fill vias with the via fill pattern specified in the color map.

The default is **off**.

For example

set via_to_layer_pattern on

write_permission

Use **set write_permission (group** [**read** | **noread**] [**write** | **nowrite**]**) (**[**public** [**read** | **noread**] [**write** | **nowrite**]**)** to control file read and write permissions when you save placement or routing information.

This condition sets the **group** and **public** permissions on files you save from SPECCTRA using the **write** command. The choices are

read or **noread**, which allows or prohibits others to view or load the file into SPECCTRA.

write or **nowrite** which allows or prohibits others to save the file.

The permissions default to your user permissions set in your login account. Changes you make with the **set write_permission** command apply only during the current SPECCTRA session.

Note

The **set write_permission** command is not available in the Windows version of SPECCTRA.

Set conditions overview

You can use the set command to set conditions that control

- Autorouting
- Routing with vias
- Rule checking
- Noise and crosstalk calculations
- Interactive routing
- Graphic display features
- Other features

These categories are used here only for convenience. For instance, some of the autorouting controls also apply to interactive routing, and some of the rule checking and noise and crosstalk controls also apply to autorouting or interactive routing.

For autorouting, you can control

- The update interval of the status file
- Whether the autorouter must route to the origins of pins
- Whether the autorouter gathers the wires of a differential pair or bundle (bus) before connecting the wires to pins
- Whether the autorouter can dynamically swap net pin connections on swappable pins
- Whether the autorouter can swap fanout connections on swappable pins
- Whether the autorouter performs pin width tapering during autorouting rather than during post-processing
- Whether fences are hard or soft
- Whether the autorouter attempts to rip up and reroute nets with order violations
- The cost of routing stubs that are longer than the **max_stub** length rule
- Whether the autorouter can route diagonal wires always, never, or only when needed for diagonal memory routing, through staggered pin arrays, and near existing diagonal wires. .
- Whether the autorouter rounds off calculations for pad location coordinates

For routing with vias, you can control

- Whether the autorouter can rotate nonsymmetrical jumper vias
- How SPECCTRA measures gap and stagger distances for blind and buried vias.

For rule checking, you can control

- Whether same net clearance violations are checked
- Whether average lengths of differential pairs are used for rule checking

For noise and crosstalk calculation, you can control

- Which crosstalk model is used for the crosstalk report
- Whether the calculated total noise accumulated on a victim net is computed as a simple sum or as the square root of the sum of squares of the noise contributions from the aggressor nets.
- Whether noise calculations are made using stairstep or linear interpolation
- Whether pin shapes are included in measurements for calculations that use parallel_noise and tandem_noise rules.
- The number of adjacent signal layers considered in tandem noise and tandem segment crosstalk calculations

For interactive routing, you can control

- What selection criteria is used for cut wire segments
- Whether the interactive router can automatically merges polygons in Move mode

- Whether the interactive router displays the snap grid cursor in interactive editing and drawing modes
- Whether the interactive router can undo pushed wires or vias when you cancel a wire edit in Edit Route mode

For graphic display features, you can control

- How layers display
- How many color gradations are used in the density analysis display
- Whether dynamic pan and zoom are enabled
- Whether shadow mode is used to display selected objects
- Whether the via fill pattern matches the layer fill pattern
- Whether SPECCTRA permits or prohibits all work area repaints, or permits repaints only after explicit viewing operations (such as zoom, pan, or repaint).
- Whether SPECCTRA performs automatic repaints after operations performed by commands in a do file.

Other conditions you can control are

- Whether SPECCTRA warns you about a user defined property it does not recognize
- What permissions are set on files you save from SPECCTRA (on UNIX platforms only)
- Whether licensed MicroVia features are available at the command line and in the GUI. (MicroVia features made available by this set command require a special license).

setexpr

The **setexpr** command evaluates an expression and stores the result in a variable.

Use the **setexpr** command to create variables (<*variable_name*>) by evaluating an <*expression*>.

Any variables you create can be used in subsequent **setexpr** expressions, and in the evaluate, if and while commands. You can redefine variables by specifying the same <*variable_name*> in the **setexpr** command.

Note

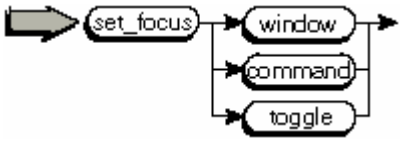
Only user-created variables can be changed with the **setexpr** command. System variables are changed only by the autorouter. See <*system_variable*> in the *Design Language Reference*.

Command examples

setexpr count (0)
setexpr count (count + 1)

set_focus

The **set_focus** command controls the focus for alphanumeric keystrokes.



Use the **set_focus** command to change the focus for alphanumeric keys that have been assigned a command function. The choices are

window, to perform a command assigned to an alphanumeric key when you move the pointer into the work area and press the key

command, to perform a command assigned to an alphanumeric key when you move the pointer into the command entry area and press the key

toggle, to toggle the focus between the work area and the command entry area.

To display a list of currently defined keys, you can use the defkey command or the Define Keys dialog box in the GUI.

Note

You can use the Tab key to toggle the focus.

Command examples

set_focus window
set_focus command
set_focus toggle

setup_check

The **setup_check** command sets checking options for the session.

<check_type>

The check types you can specify with the setup_check and check commands are:

Type	Defaults to
conflict	on
length	on
limit_way	off
max_vias	off

miter	off
order	off
pin	off
polygon_wire	off
protected	off
same_net_check	off
stagger	off
stub	off
use_layer	off
use_via	off
xtalk (crosstalk)	on

By default, the routing checker detects routing conflicts and violations of crosstalk and length rules. The <*check_type*> options for this command turn **on** or turn **off** these default settings and several others.

After you use **setup_check** to turn on the options you want to check and turn off those you do not want to check, you must use the **check** command to perform the rules check.

The checking options you set remain in effect only during the session in which you set them. They revert to the default settings at the start of each session.

You can override any of the checking options for a single execution of the **check** command without using **setup_check**. See the check command and the include option for more information.

Note

Checking options that are **on** by default directly affect the operation of the autorouter when set to **off**. For example, the autorouter normally checks for and eliminates conflicts. If you use the **setup_check** command and set conflict checking to **off**, the router does not eliminate routing conflicts.

See also

check
set

Command examples

setup_check (miter on) (polygon_wire off)

setup_check (use_layer on) (use_via on) (limit_way on) (same_net_check on)

<check_type>

conflict

Checks for shorts and clearance violations.

The default is **on**.

length

Checks for violations of length rules.

The default is **on**.

limit_way

Checks for violations of the **rule** command limit_way rule.

The default is **off**.

max_vias

Controls whether the maximum via rules for nets, classes, groups, and fromtos are checked. The default setting for this control is **off**, which means maximum via rules are not checked. See the **rule** command for setting max_vias rules.

miter

Checks for unmitered wire corners.

The default is **off**.

order

Checks routed wiring for violations of the net ordering rules, and highlights violations in the work area when you run the **check** command.

Note

You might not want to turn on both **order** and **stub** at the same time because the violations appear similar when highlighted in the work area.

pin

Checks for clearance violations between pins and other objects.

The default is **off**.

polygon_wire

Checks for clearance violations between wiring polygons and other objects.

The default is **off**.

protected

Checks for clearance violations between protected wires or vias and other objects.

The default is **off**.

same_net_check

Checks for clearance rule violations between objects on the same net. A same net clearance rule violation occurs when a wire segment, via, or pin is too close to another object on the same net.

The default is **off**.

Note

The via_via and via_via_same_net clearance rules are always checked and are not affected by this control. Only clearance rules, which are used to prevent unintended shorts, are checked.

stagger

Checks for violations of the **rule** command maximum stagger rule.

The default is **off**.

stub

Checks for violations of the **rule** command max_stub length rule, and highlights violations in the work area when you run the **check** command.

The default is **off**.

Note

You might not want to turn on both **stub** and **order** at the same time because the violations appear similar when highlighted in the work area.

use_layer

Checks for violations of the **circuit** command use_layer rule.

The default is **off**.

use_via

Checks for violations of the **circuit** command use_via rule.

The default is **off**.

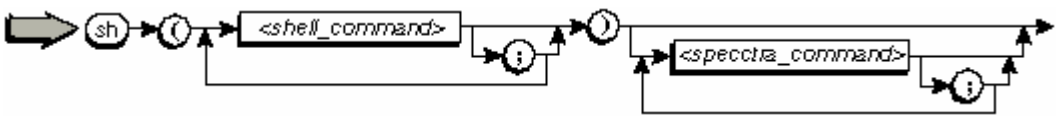
xtalk

Checks for violations of crosstalk rules.

The default is **on**.

sh

The **sh** command runs system-level commands from inside the SPECCTRA GUI.



The **sh** command lets you use shell commands in UNIX or DOS commands and file executables in Windows by entering them in the SPECCTRA command entry area. To enter multiple shell commands, separate the shell commands with a semicolon (;).

If you enclose shell commands within parenthesis, you can include SPECCTRA commands on the same line.

If a command starts another application, you must close the application window before you can use SPECCTRA. If it does not open a separate window, the command output goes to the output window.

Notes

A shell command in Windows NT is a command that you can enter at the command prompt.

Command examples

```
sh calc
sh ls -l *.w
sh more monitor.sts
sh ps -aux
sh date > routing.note ; vi routing.note
sh uncompress revb.wir.Z
(sh telnet); route 5; clean 2
```

shield

The **shield** command automatically routes shield wires around existing wires.

Use the **shield** command to route shields around unshielded wires of nets that have a shield rule. This command is useful in cases where you need to shield wires that contain t-junctions.

Before using the **shield** command, make sure that there is sufficient clearance around unshielded wires to route the shield wires. You can do this by increasing the clearance of these nets before routing them. After those nets have been routed with extra clearance, restore the previous clearance for those nets and use the **shield** command to route the shields. The command routes shields for all nets that have a shield rule but no shield wire(s).

Notes

If you select wires before using the **shield** command, the command routes shields only for the selected wires that have a shield rule.

The **shield** command does not provide clearance checking. To avoid creating clearance violations when you use this command, increase the clearance rule for the nets before you route them initially. Be sure to restore the previous clearance for the nets after you route them and before you use the **shield** command.

See also

shield rule for the circuit command

shield rules for the rule command:

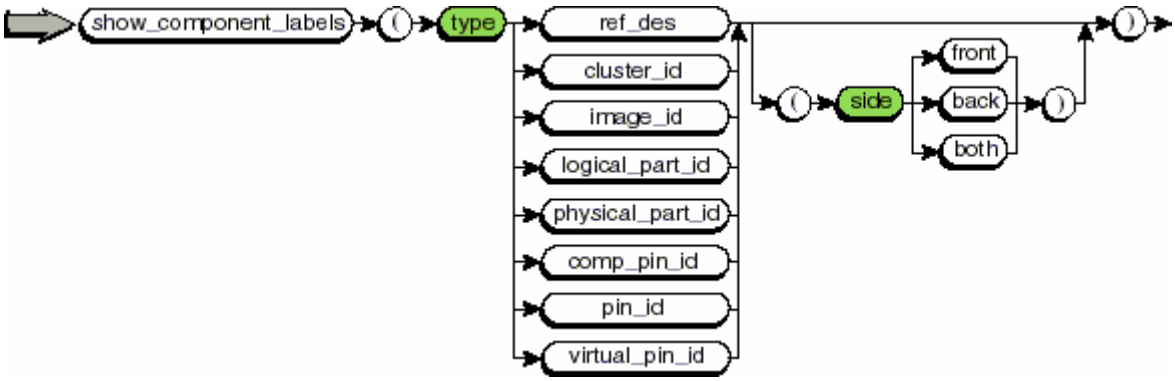
Command example

The following example routes shields for four nets that will contain t-junctions after routing. To ensure that there is adequate clearance for the shields, the clearance rule for the nets is increased before autorouting. The design is then routed. Because the four nets contain t-junctions, shields are not routed, even though they have a shield rule. Finally, the nets are selected again; their clearance rule is restored; and the shield command is executed to add the shields.

```
select net net1 net2 net3 net4
rule selected (clearance 10)
unselect all
route 25
select net net1 net2 net3 net4
rule selected (clearance 5)
shield
```

show component_labels

The **show component_labels** command controls which object identifiers appear when component labels are turned on for viewing.



type

Controls which identifiers you want to display in the component labels. The choices are

ref_des displays component reference designators.

cluster_id displays cluster names. Each component in a cluster displays the name of the cluster.

image_id displays image names.

logical_part_id displays the logical part names for components mapped by the <*logical_part_descriptor*> in the design file.

physical_part_id displays the physical part names for components mapped by the <*physical_part_descriptor*> in the design file.

comp_pin_id displays component reference designators and image pin names.

pin_id displays image pin names.

virtual_pin_id displays virtual pin names.

side

Controls whether the current operation applies only to the front side (**front**), back side (**back**), or both sides (**both**) of the PCB. The default is **both**.

Use this command to specify the object identifiers that appear in component labels. You can display labels for components, component clusters, images, logical parts, physical parts, pins or virtual pins. You can also control whether to display labels for components on the front side, the back side, or both sides of the PCB.

The **show component_labels** command controls only which labels are displayed. The vset command controls whether labels are visible or hidden. At the beginning of a session, component labels are hidden by default.

The **show component_labels** command does not automatically repaint the screen. Use repaint to update the screen display.

Command examples

vset component_labels on
show component_labels (type ref_des)
repaint

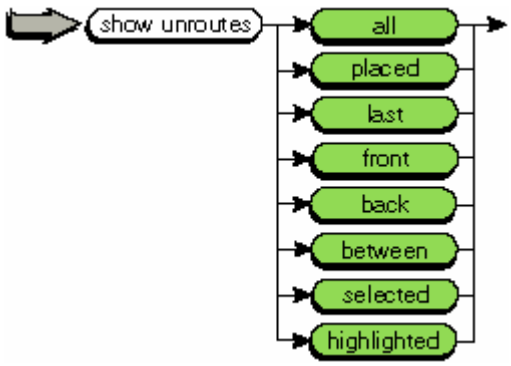
show component_labels (type image_id)
repaint

show component_labels (type comp_pin_id) (side back)
repaint

show component_labels (type physical_part_id) (side front)
repaint

show unroutes

The **show unroutes** command controls which guides (sometimes called unrouted connections or unroutes) are displayed.



all

Displays all unrouted connections

placed

Displays only those guides that identify unrouted connections to components placed within the placement boundary.

last

Displays only those guides that identify unrouted connections to the last component placed during the most recent automatic placement operation.

front

Displays only those guides that identify unrouted connections to components on front side of PCB.

back

Displays only those guides that identify unrouted connections to components on back side of PCB.

between

Displays only those guides that identify unrouted connections between pins on the front side of the PCB and pins on the back side.

selected

Displays only the selected guides or guides connected to the selected components, wires, nets, or pins.

highlighted

Displays only those guides that are currently highlighted.

Use this command when you want to display only a subset of the guides for unrouted connections.

Guides can obscure other objects in densely populated designs. By viewing guides selectively you can reduce the complexity of the display. For example, you can display just the guides for all placed components, the last component placed, components on one side of the PCB, or components that are highlighted.

The **show unroutes** command controls only which guides are displayed. The vset command controls whether guides are displayed or hidden. At the beginning of a SPECCTRA session, all guides are displayed by default.

The **show unroutes** command does not automatically repaint the screen. Use repaint to update the screen display.

Command examples

vset unroutes on
show unroutes placed
repaint

show unroutes highlighted
repaint

show unroutes front
repaint

show unroutes selected
repaint

skill_cmd

The **skill_cmd** command allows you to issue SKILL commands from the SPECCTRA command entry area, without changing to SKILL mode (see skill_mode).

Command examples

skill_cmd(printf(“total components = %d\n” totalcomp))
skill_cmd(load(“~design_macros/sum_comps.il”))

See also

skill_mode
cct_mode

skill_mode

The **skill_mode** command sets the SPECCTRA command entry area to accept SKILL programming language commands (SKILL mode). To change the command entry area to accept SPECCTRA commands, you must issue the **cct_mode** command.

After entering the **skill_mode** command, you can type SKILL commands in the command entry area. You can also use the SKILL **load** command to execute a file with SKILL commands.

The following SPECCTRA system variables are available for use by SKILL.

While executing SKILL commands in SKILL mode, you can use the **cct_cmd** command to execute SPECCTRA commands.

Command examples

skill_mode
printf(“total components = %d\n” totalcomp)
cct_mode

skill_mode
load “~design_macros/sum_comps.il”
cct_mode

skill_mode
for (i 0 5{cct_cmd(“z out”)})
cct_mode

See also

cct_cmd
cct_mode
skill_cmd

System variables

The following SPECCTRA system variables are available.

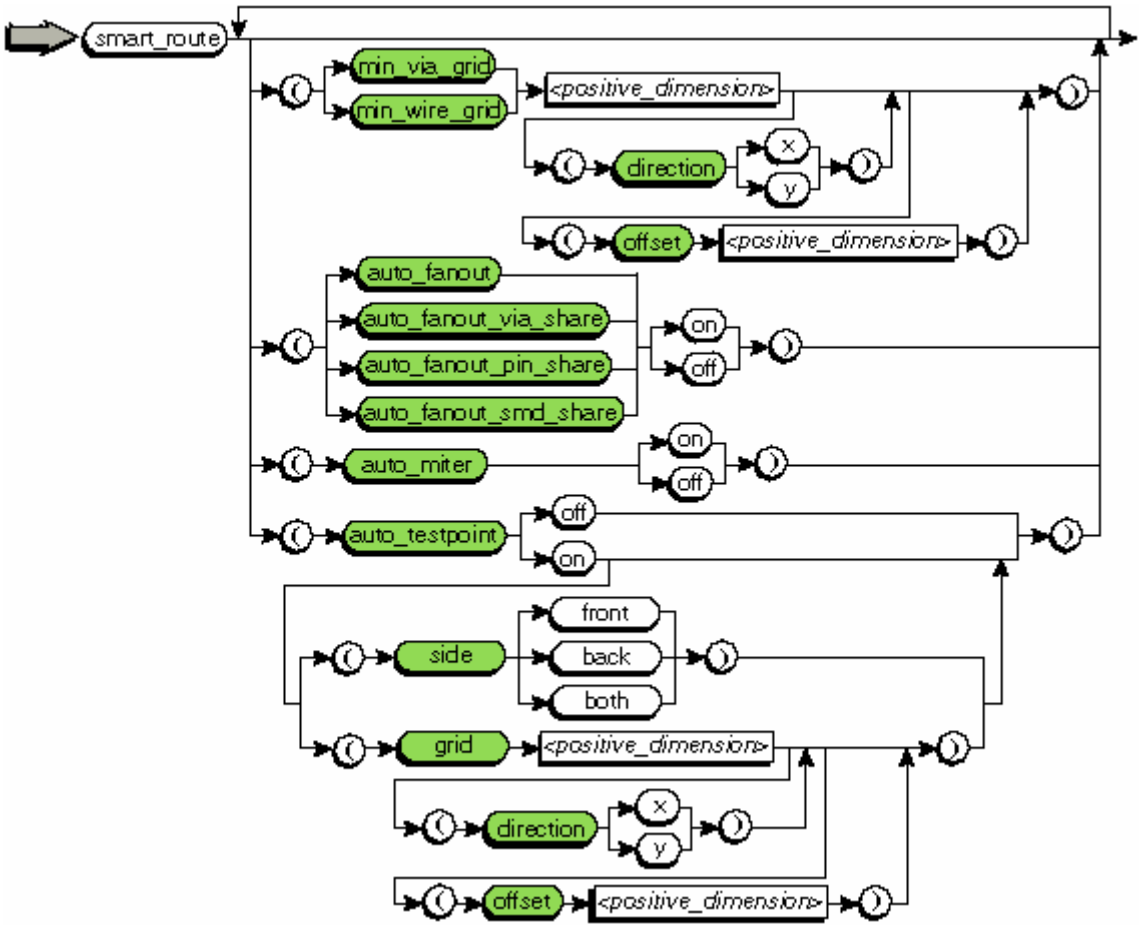
Variable Name	Type	Definition
bottom_layer_sel	Integer	1 if bottom layer is selected, 0 if not selected.
complete_wire	Number	Completion ratio expressed as a percentage.
conflict_clearance	Integer	Number of clearance rule violations.
conflict_crossing	Integer	Number of crossing conflicts.
conflict_wire	Integer	Number of crossing and clearance conflicts.
conflict_xtalk	Integer	Number of crosstalk rule violations.
connections	Integer	Total number of connections to be routed.
current_wire	Integer	Current wire being routed or rerouted.
locked_comp	Integer	Number of locked components.
partial_selection	Integer	Value equals 0 if no nets or all nets are selected; value equals 1 when one or more nets but fewer than all nets are selected.
placedcomp	Integer	Number of placed components.

power_layers	Integer	Number of power layers.
reduction_ratio	Integer	Conflicts reduction ratio from last completed routing pass.
reroute_wire	Integer	Number of wires and wire segments to be rerouted in the current pass.
route_pass	Integer	Current routing pass or last pass.
sel_comps_list	String	Names all selected components.
sel_nets_list	String	Names of all selected nets.
sel_signal_layers	Integer	Number of selected signal layers.
selectedcomp	Integer	Number of selected components.
selectednet	Integer	Number of selected nets.
signal_layers	Integer	Number of signal layers.
smd_pins	Integer	Number of SMD pads.
thru_pins	Integer	Number of through-hole pins.
top_layer_sel	Integer	1 if top layer is selected, 0 if not selected.
total_pass	String	Total passes for the current command.
total_pins	Integer	Total number of pins.
total_vias	Integer	Total number of vias.
totalcomp	Integer	Total number of components on the PCB.
unconnect_wire	String	Unconnected wires (unconnects).
units	String	Unit of measure set by user.
unplaced_comp	Integer	Number of unplaced components outside the

		placement boundary.
unplaced_large	Integer	Number of large components outside the placement boundary.
unplaced_small	Integer	Number of small components outside the placement boundary.

smart_route

The **smart_route** command autoroutes your design based on how your design is converging.



min_via_grid

Sets the minimum X and Y via grid (<*positive_dimension*>). You can

Specify a value for only the X or Y axis (**direction**).

Specify an offset value for the uniform X and Y grid (**offset**).

The default is the via grid set in the design file.

min_wire_grid

Sets the minimum X, Y wire grid (<*positive_dimension*>). You can

Specify a value for only the X or Y axis (**direction**).

Specify an offset value for the uniform X and Y grid (**offset**).

The default is the wire grid set in the design file.

direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

auto_fanout

Controls whether the autorouter preroutes SMD pads with a short wire and via. The choices are

on turns on **auto_fanout**, which means the autorouter preroutes escape wires and vias.

off turns off **auto_fanout**, which means the autorouter does not preroute escape wires and vias.

The default is **on**.

auto_fanout_via_share

Controls whether SMD pads on the same net can share escape vias when **auto_fanout** is **on**. The choices are

on turns on **auto_fanout_via_share**, which means SMD pads can share the same escape via.

off turns off **auto_fanout_via_share**, which means SMD pads connect to unique escape vias.

The default is **on**.

auto_fanout_pin_share

Controls whether SMD pads can escape to through-pins when **auto_fanout** is **on**. The choices are

on, turns on **auto_fanout_pin_share**, which means SMD pads can escape to through-pins if the cost is lower than the cost to use a via.

off turns off **auto_fanout_pin_share**, which means SMD pads escape to vias only.

The default is **on**.

auto_fanout_smd_share

Controls whether the autorouter routes connections between nearby SMD pads on

the same net so that they share an escape wire and pin or via when **auto_fanout** is **on**. The choices are

on turns on **auto_fanout_smd_share**, which means connections between SMD pads are routed so that they share the same escape wiring if the cost is lower than the cost to use escape wires and pins or vias for each SMD pad.

off turns off **auto_fanout_smd_share**, which means each SMD pad connects to an escape wire and pin or via.

The default is **off**.

auto_miter

Controls whether the autorouter does mitering after all **route**, **testpoint**, and **clean** passes are completed. No mitering is done if the routing is not 100 percent. The choices are

on turns on **auto_miter**, which means the autorouter changes corners from 90 to 135 degrees.

off turns off **auto_miter**, which means the autorouter does not change 90 degree corners.

The default is **off**.

auto_testpoint

Controls whether the autorouter adds test points. The choices are

on turns on **auto_testpoint**, which means the autorouter adds test points for routed signal nets using the **side** and **grid** settings.

off turns off **auto_testpoint**, which means the autorouter does not add test points

The default is **off**.

side

Identifies the test point probing layer as the top (**front**), bottom (**back)**, or both top and bottom (**both)** sides of the PCB.

The probing layer contains exposed test vias (not covered by a component body).

The default is **back**.

grid

Defines a uniform grid or nonuniform X and Y grids. Grids can be offset. You can

Specify the grid value (<*positive_dimension*>)

Specify an X or Y direction (**direction**)

Specify an offset (**offset**)

If you want a uniform grid, do not specify a direction.

The default test point grid is the current pcb via grid. The grid for test point insertion is a probing grid that should match your bed-of-nails tester.

direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

You can use the **smart_route** command to evaluate your design and run autorouting commands that produce the best possible completion. This command adjusts autorouting based on the conflict reduction rate, the routing completion, the number of failures, and the number of layers. You can start the **smart_route** command at any stage of routing completion.

The routing progress indicator monitors and displays the progress of the **smart_route** command using a traffic light icon. You can click on the icon to display detailed information in a dialog box.

When you use **smart_route** you can

- Set the minimum via grid and minimum wire grid
- Preroute short escape wires from SMD pads to vias (fanout)
- Change corners from 90 to 135 degrees (miter)
- Add test points

When **auto_fanout** is on, the fanout operation is activated if there are more than two signal layers, or if the top or bottom layer is not selected for routing. You can set controls that allow the autorouter to

- Share escape vias for more than one SMD pad on the same net
- Escape to through-pins if the cost is lower than the cost to use a via
- Route connections between nearby SMD pads so that they share the same escape wire and pin or via if the cost is lower than the cost to use separate escape wires, pins or vias

When **auto_miter** is on, the miter operation is activated after all **route**, **testpoint**, and **clean** passes are completed. No mitering is done if the routing is not 100 percent.

When **auto_testpoint** is on, the test point operation is activated when the router reaches 80 percent completion or at the end of the forced convergence loop. The forced convergence loop occurs when **smart_route** adjusts internal costs and attempts to force convergence (routed 100 percent) by routing small blocks of routing passes. You can

- Identify the probing layer side as **front**, **back**, or **both**. The probing layer is the layer on which test vias are exposed (not covered by a component body). You can specify separate test point rules for the front or back sides of the design.
- Set the grid for test via insertion (should match your bed-of-nails tester).

When you set the minimum via grid, minimum wire grid, or auto test point grid, you can specify a uniform grid or nonuniform X and Y grid. You can specify offsets.

Notes

You can route selected nets with the **smart_route** command. See also the select command.

The **smart_route** command automatically enables the bestsave function, creating a wires file with the default filename *bestsave.w*. You can specify a different filename by using the bestsave command.

For more miter and test point options than are available in **auto_miter** and **auto_testpoint**, see the miter and testpoint rule commands.

See also

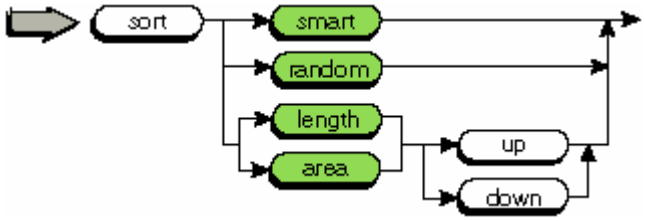
fanout
grid commands

Command examples

```
smart_route
smart_route (min_via_grid 1) (min_wire_grid 1)
smart_route (min_via_grid 5 (direction x)) (min_wire_grid 1 (offset 5))
smart_route (auto_fanout off)
smart_route (auto_miter on)
smart_route (auto_testpoint on (side back) (grid 25))
```

sort

The **sort** command controls how connections are scheduled for autorouting.



smart

Sorts unrouted fromtos according to a priority scheme derived from layout parameters.

random

Sorts unrouted fromtos according to a random, or deliberately chance, order.

length

Sorts unrouted fromtos according to their Manhattan lengths (Dx + Dy). Sorting is either short-to-long (**up**) or long-to-short (**down**).

The default is **up**.

area

Sorts unrouted fromtos by the size of the area that contains the fromtos. Sorting is either small-to-large (**up**) or large-to-small (**down**).

The default is **up**.

SPECCTRA sorts unrouted fromtos prior to each autorouting pass. You can use the **sort** command to specify the sorting method.

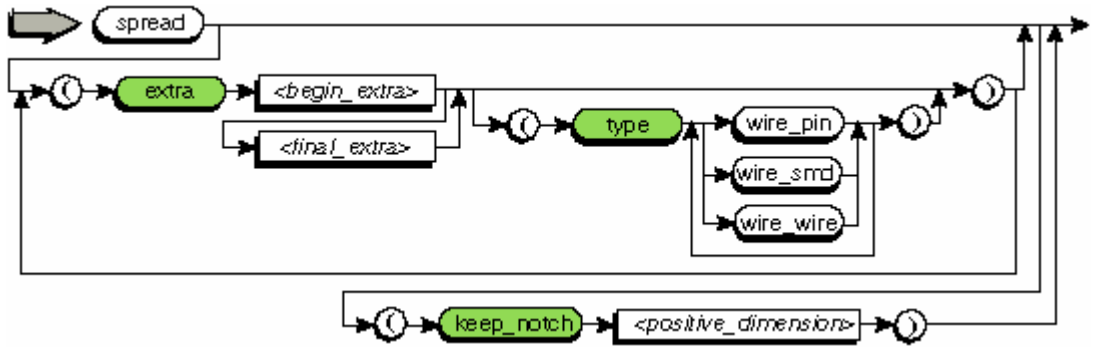
If you do not use the **sort** command, the autorouter uses the **smart** sorting method. When you have a large number of long diagonal fromtos, you can force the autorouter to route them first by using **length** sorting method with the **down** keyword before you run any **route** commands .

Command examples

sort length down
sort area up

spread

The **spread** command attempts to add space between wires, and between wires and pins.



extra

Adds extra wire-to-object clearances. You must specify initial extra clearance (<*begin_extra*>). Optionally, you can specify the last extra clearance (<*final_extra*>) attempted, which causes **spread** to function in a progressive mode. These values must be positive real numbers.

If <*final_extra*> is not supplied, <*begin_extra*> is the only value tried.

type

Specifies the object-to-object type that the extra clearance is applied to. The choices are:

wire_pin, which means wires and through-pins.

wire_wire, which means adjacent wires.

wire_smd, which means wires and SMD pads.

keep_notch

Specifies the minimum U-type notch (<*positive_dimension*>) allowed.

The default dimension is the wire-to-wire clearance value.

The **spread** command adds extra wire-to-object clearances to improve manufacturability of a printed circuit board. This command repositions wires to create extra clearances between wires and pins, wires and SMD pads, and adjacent wire segments. The **spread** command does not move or remove vias. If **type** is not specified, extra clearances are attempted for all types.

The critic command can remove U-type notches that are sometimes created when **spread** adds extra clearance between wires and pins. To retain the extra space (and not remove the notch) use the **keep_notch** option.

The **spread** command does not introduce new conflicts.

If you enter both **extra** values, and <*begin_extra*> is smaller than <*final_extra*>, the two values are automatically swapped. When you use the **spread** command without options it is equivalent to entering the command

```
spread (extra <begin_extra>)
```

where <*begin_extra*> defaults to one-half of the current clearance rule values for **wire_pin, wire_smd,** and **wire_wire**.

When you specify both <*begin_extra*> and <*final_extra*>, multiple passes are invoked that use progressively smaller values to create extra wire-to-object clearances. In this progressive mode of operation, <*begin_extra*> is the first extra clearance value attempted.

After the first pass, the <*begin_extra*> value is divided by two, and that value is attempted for the next pass. This process continues until the divide-by-two operation results in a value equal to or less than <*final_extra*>, or until five passes elapse. If the divide-by-two operation results in a value less than <*final_extra*>, the final pass is invoked and the <*final_extra*> clearance value attempted. If after four passes the divide-by-two result is greater than <*final_extra*>, that divide-by-two value is used for a fifth and final pass.

When you use the progressive mode with a wire grid, the grid should be smaller than the amount of additional clearance you want to add. During the progressive mode, if a divide-by-two operation results in a value that is smaller than the defined wiring grid the function terminates.

When just one or two clearance types are specified, **spread** is applied only to the specified types. The unspecified type is excluded. Extra clearance values apply only during the **spread** operation. When the command finishes, clearance rules revert to their default or previously specified values.

Use **spread** after completion of all route and clean passes, and before you use miter or recorner commands.

Notes

The **spread** command does not follow FST rules.

Command examples

spread
spread (extra 3 (type wire_wire wire_pin))
spread (extra .1)
spread (extra 40 5)
spread (extra 5 (type wire_wire)) (extra 6 (type wire_smd))
    (extra 8 2 (type wire_pin))
spread (keep_notch 12)

status_file

The **status_file** command redirects the routing status information from the default monitor.sts file to the filename you specify.

During autorouting operations, SPECCTRA automatically saves status information in the file *monitor.sts* in the same directory as the design file. If you want to rename this file and save it in a different directory, use the **status_file** command to specify the file and directory. For general information about file naming, see file naming conventions.

You can also redirect the monitor.sts file by using the -s switch when you start SPECCTRA.

See also **update_interval** in the set command.

Command examples

status_file grid1.sts

stop

The **stop** command terminates a paused autorouting or placement operation.

The **stop** command terminates an active autorouting or placement operation and places the system in ready mode. For example, if you start a route pass, you can type **stop** to terminate the pass and return to ready mode. This is the same as clicking Pause, and then clicking Stop in the GUI.

When you issue a routing or placement command from a do file, you can enter **stop** and all subsequent commands in the do file are ignored. If the do file is started with the -do switch when you start SPECCTRA, and you also specify -quit, **stop** terminates the autorouter and exits.

You can use the **stop** command during the following routing operations:

clean
critic

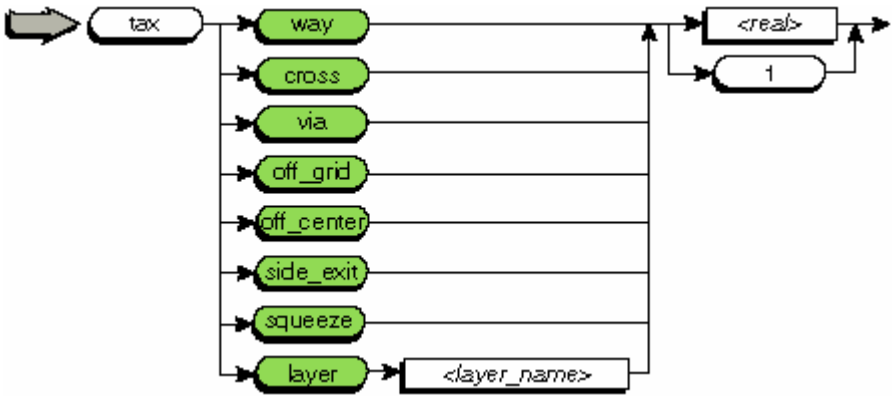
fanout
filter
miter
route
smart_route
spread

You can use the **stop** command during the following placement operations:

autodiscrete
autorotate
form_cluster
initplace
interchange
swap

tax

The **tax** command applies a factor to adjust the autorouter costs.



way

The cost to route in the wrong direction. For example, the cost of horizontal wire segments routed on a vertical layer.

cross

The cost of a crossing conflict.

via

The cost to use a via.

off_grid

The cost to enter or exit a pin off grid.

off_center

The cost to enter or exit a pin off center.

side_exit

The cost to exit pins on the long side.

squeeze

The cost to create a wire-to-via clearance violation.

layer

The cost to use a layer (<*layer_name*>) for routing.

Use the **tax** command to control autorouting costs by applying a multiplier (<*real*>) to the autorouter's internal cost parameters. The cost parameters are represented by the **way**, **cross**, **via**, **off_grid**, **off_center**, **side_exit**, **squeeze**, and **layer** keywords.

For example, **tax way .9**, multiplies the autorouter's internal wrong-way cost by 0.9. The autorouter uses this altered value until the internal parameter changes. The taxing factor is then re-applied to alter the new internal value. You can also control autorouting costs by using the cost command, but its cost specifications remain fixed and in effect until you change them.

The default factor value for the **tax** command is 1. You can reset to this value at any time.

Both **cross** and **squeeze** impact the number of conflicts and the number of unconnected wires. If **squeeze** and **cross** are less than 1.0, the autorouter generates more conflicts but fewer unconnects. Conversely, if these parameters are greater than 1.0, the autorouter generates fewer conflicts and potentially more unconnects.

See also the route command.

Note

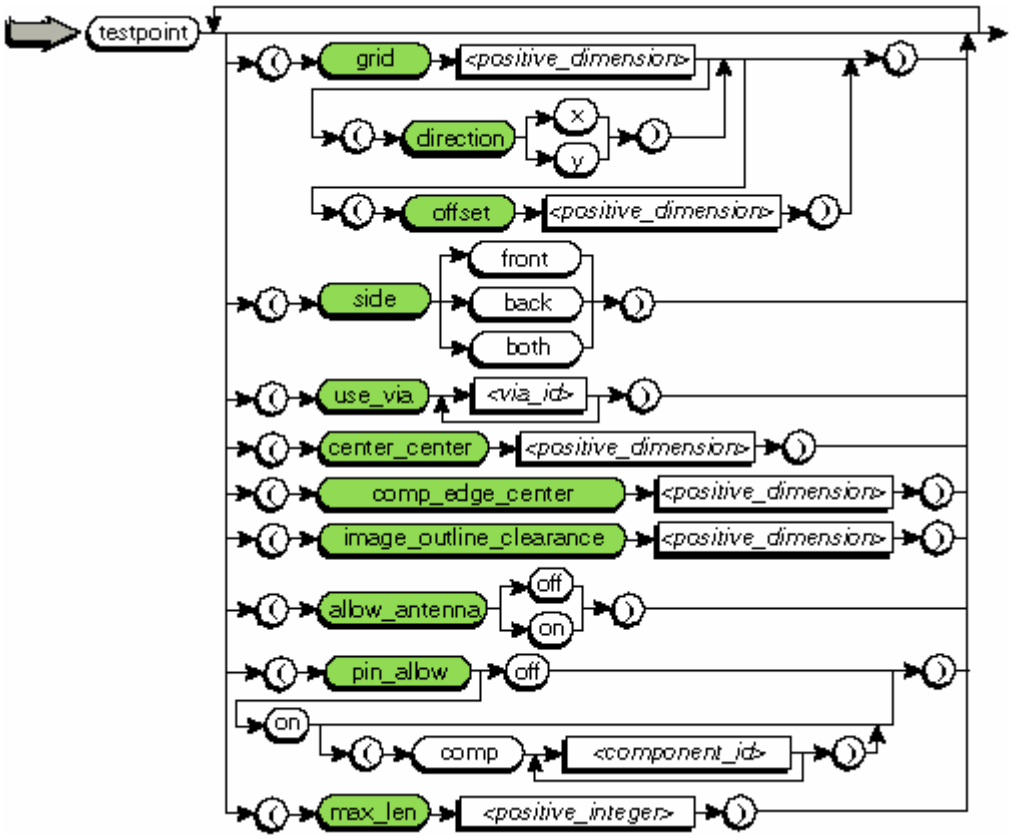
SPECCTRA maps the multiplier (<*real*>) you specify for **tax layer** to an internal costing curve before applying the command. The valid range for the <*real*> value is from 1 to 11. A value greater than 11 is mapped the same as 11. A value less than 1 is ignored.

Command examples

```
tax cross .9
tax via .8
tax layer S1 1.1
```

testpoint

The **testpoint** command controls test point insertion.



grid

Defines a uniform grid or nonuniform X and Y grids. Grids can be offset. You can

Specify the grid value (<*positive_dimension*>)

Specify an X or Y direction (**direction**)

Specify an offset (**offset**)

If you want a uniform grid, do not specify a direction.

The default test point grid is the current pcb via grid. The grid for test point insertion is a probing grid that should match your bed-of-nails tester.

direction

Specifies an **X** or **Y** grid. If **direction** is not set, the grid is a uniform X and Y grid.

offset

Specifies an offset (<*positive_dimension*>) for the X and Y grids.

side

Identifies the test point probing layer as the top (**front**), bottom (**back)**, or both top and bottom (**both)** sides of the PCB.

The probing layer contains exposed test vias (not covered by a component body).

The default is **back**.

use_via

Identifies one or more via padstacks (<*via_id*>) to be used as test points.

If no **use_via** value is specified, the autorouter uses the smallest size via that spans all layers and is selected for routing.

center_center

Controls the minimum distance (<*positive_dimension*>) permitted between the centers of any two test points.

If the **center_center** rule is different for two test points, the larger value is used.

If no value is given, center-to-center test point checking is not done.

comp_edge_center

Controls the minimum distance (<*positive_dimension*>) permitted between any test point center and a component boundary edge.

If no value is given, center-to-component edge checking is not done.

image_outline_clearance

Controls the minimum distance (<*positive_dimension*>) permitted between any test point edge and a component boundary edge.

The default is the area-to-testpoint object-to-object clearance specified in the clearance rule.

allow_antenna

Controls whether antennas (stubs) are permitted when test points are added. Antennas are allowed when this rule is **on**.

The default is **on**.

pin_allow

Controls whether through-pins can be used as test points.

When on, you can use **comp** (<*component_id*>) to identify a list of components with through-pins that can be used as test points. If a component list is not included, all through-pins that meet grid and clearance requirements are used.

The default is **off**.

max_len

Restricts the routed length of testpoint antennas. The length is measured from a pad’s origin to the center of the testpoint via.

use_rules

Specifies that the **testpoint** command follow pcb level test point rules that you set using the <*testpoint_rule_descriptor*> in the **rule** command.

You can use the **testpoint** command to improve PCB testability by adding test points to routed signal nets as a post-processing operation.

A test point is a through-pin (pin) or via that SPECCTRA marks as a test point because a **testpoint** control is set for the net that contains the pin or via. A test via can be a plated-through type or a single surface pad. When an exposed via (not covered by a component body), is not available, SPECCTRA pushes the existing via to an available test point grid site. If this fails, SPECCTRA adds an additional test point via.

Use the **testpoint** command to set the following controls:

- Specify the grid used for placing test vias. The default is the current PCB via grid.
- Identify the probing layer **side** for test points as **front**, **back**, or **both**. You can specify separate **testpoint** rules for the front or back sides of the PCB.
- Specify one or more via padstacks to be used as test points. If you do not use this control, the autorouter chooses a via. Single layer padstacks can be used as test vias.
- Control the minimum center-to-center distance between test points.
- Control the minimum distance between the center of the test point and the component edge (boundary).
- Control the minimum distance between the edge of the test point and the component edge (boundary). You can specify only one **image_outline_clearance** value.
- Control whether antennas (stubs) are allowed.
- Control whether through-pins are used as test points. You can identify a list of components with through-pins that can be used as test points. If a component list is not included, all through-pins that meet the grid and center-to-center requirements are used.
- Control the maximum length of a connection between a net and an inserted test point via.

When you set the minimum test point grid, you can specify a uniform grid or nonuniform X and Y grids. You can specify offsets.

The test point environment is established by the last **testpoint** command. Any environment settings established by a previous **testpoint** command are overridden by the next command.

Using the testpoint command

You should run the **testpoint** command after all routing is completed, but before you

use clean, spread, and miter commands. When used at this stage, the operation takes advantage of existing vias.

A common method for achieving improved testability is to escape all SMD pins and then protect the SMD-to-via connections in order to guarantee that all SMD pins have a via for testing. This method can be useful for autorouting multilayer designs, but might be wasteful when compared to the autorouter's test point method. Consider the following factors:

- Many extra vias are required for those connections that can otherwise be completed without a via by wiring directly on the SMD layer.
- When vias are protected, the ability to rip-up, reroute, and eliminate them is lost.
- The ability to route on the SMD layers is constrained by all the protected connections.
- On a five-pin net, five vias are generated and protected, but only one via is required per net for a test point.

Notes

To assign test point rules by net, class, or for the entire design, and add the test points during the next **route**, **clean** or **filter** pass, see the testpoint rule.

If you include the **use_rules** keyword, the **testpoint** command follows pcb level test point rules that you set using the **testpoint rule**. Otherwise, the **testpoint** command overrides test point rules set at the pcb level of the rule hierarchy. For example, if you enter the **testpoint** command without options, the operation proceeds with the **testpoint** command default settings, and ignores any rules set at the pcb level with the **testpoint rule**. Rules set at the higher levels are not affected.

The clearance rule controls object-to-object clearances for test points, which are edge-to-edge clearances. Special clearances, such as **center_center** and **comp_edge_center** are part of the **testpoint** command itself and are test point center checks. Test point center checking is a separate checker pass.

The smart_route command does not activate test point insertion until routing is 80 percent complete. You set the appropriate **testpoint** controls and then run **smart_route** with the **auto_testpoint** option.

The report testpoint command generates test point summary information. The test point report includes a list of nets that have no **testpoint** control in effect and those that do have a **testpoint** control for which SPECCTRA cannot find a test via site. Since the test point feature is disabled for differential pairs, you can also see a list of missing test points for differential pairs in this report.

You can add testpoints to specific nets and wires by using the select net command.

See also the delete command to delete all the test points in a design, including any dangling wiring left by the deletion of a via.

Command examples

testpoint
testpoint (side both)
testpoint (grid 0.100) (use_via V1_9 V9) (pin_allow on)

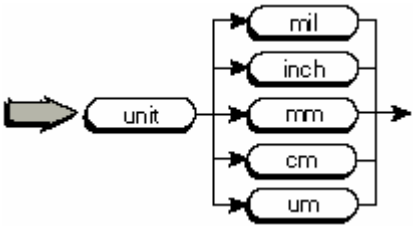
testpoint (center_center 0.100)
testpoint (image_outline_clearance 0.050)

If you want to set different test point controls for the front and back of the PCB, use separate **testpoint** commands. For example

testpoint (side front) (use_via V1-6 V1-1)
testpoint (side back) (use_via V1-6 V6-6)

unit

The **unit** command sets your working units.



You can use this command to change your working units at any time during an autorouting session. Command input, report file output, and display output are always scaled for the current working units.

The working units are:

- cm (centimeter)
- inch (decimal)
- mil (thousandth of an inch)
- mm (millimeter)
- um (micron)

Command examples

unit mil

unmiter

The **unmiter** command removes 135 degree wire corners.

You can use the **unmiter** command to remove all 135 degree wire corners. If you want this function applied to certain layers only, use **layer** and specify <*layer_id*>. The **unmiter** command does not remove round corners.

If you created 135 degree corners using the **miter** command and you must make engineering changes to your design, you should remove the 135 degree corners by using **unmiter** before you save the wires. The autorouter is more efficient when it is rerouting orthogonal wires. If you saved a wires file before you used **miter**, you do not need to use **unmiter**.

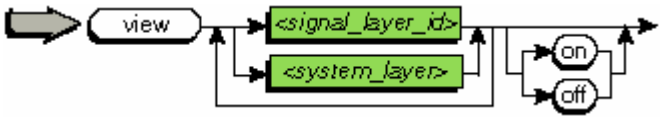
See also the miter command.

Command examples

unmiter
unmiter layer L1

view

The **view** command controls the display of layers in the SPECCTRA work area.



<signal_layer_id>

The name of a signal layer defined in the design file.

<system_layer>

The name of a system layer in SPECCTRA. Each system layer provides a visual feature such as guides or component labels.

Use the **view** command when you want to immediately change layer visibility in the SPECCTRA work area. A <*signal_layer_id*> is the name of a signal layer defined in the design file and a <*system_layer*> is the name of a system layer in SPECCTRA. Each system layer provides a visual feature such as guides or component labels.

You can specify one or more keywords separated by a space in a single **view** command. After using one or more **view** commands to turn on or turn off signal layers and system layers, the screen is automatically repainted.

The following keywords represent the system layers that you can view for routing and placement:

component_labels
error
grid
keepout
origin
pin
power
power_pins
region
site
unroute
via
via_grid
wire

If you use **view** to display component labels or routing guides (unroutes), you can use the **show** command to control what kind of labels or guides you want SPECCTRA to display. You can:

- Use the show component_labels command to display component names (reference designators), pin IDs, component names with pin IDs, component cluster names, image names, logical or physical part IDs, or virtual pin IDs.
- Use the show unroutes command to display guides for all unrouted nets, guides for unrouted signal nets connected to placed or selected components, or guides for unrouted power nets.

Notes

Additional placement keywords that represent system layers that you can view are:

place_error
place_back
place_front
place_grid

You can control the density, histogram, and force_vector displays by using these keywords with the **view** command, or by choosing commands from the Autoplace menu.

You can also control the viewing options in the Layers panel.

See also

view grid
vset

Command examples

view L2 L3 off
view L2 L3 on
view keepout on
view via on

<system_layer>

component_labels

Displays or hides labels that identify components, pins, virtual pins, images, logical parts, physical parts, virtual pins, or component clusters, depending on the which label **type** you choose with the **show component_labels** command.

error

Displays or hides routing rule conflict and violation symbols for all visible signal layers.

grid

Displays or hides the wire grid if defined. You can use the **view grid** command to

control whether the wire grid is displayed as dots or lines.

keepout

Displays or hides keepout areas on all visible signal layers.

origin

Displays or hides component origins (when component outlines are visible) on all visible signal layers.

pin

Displays or hides component pins (when component outlines are visible) on all visible signal layers.

power

Displays or hides guides (flight lines) that show unrouted power net connections.

power_pins

Displays or hides power (P) and ground (G) labels on power pins when component outlines are visible.

region

Displays or hides region boundaries on all visible signal layers.

site

Displays or hides the placement sites of selected images.

unroute

Displays or hides guides (flight lines) that show unrouted pin-to-pin connections.

via

Displays or hides vias for all visible signal layers.

via_grid

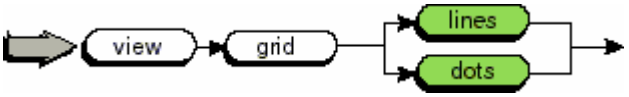
Displays or hides the via grid if defined.

wire

Displays or hides routed wires for all visible signal layers.

view grid

The **view grid** command changes the display of the wire and placement grids.



lines

Displays the grid as lines.

dots

Displays the grid as dots.

The grid displays as lines or dots. The default is lines.

See also

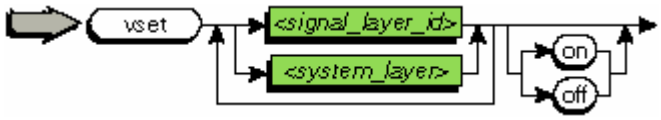
grid place
grid_wire_cmd

Command Example

view dots

vset

The **vset** command presets the layer display in the SPECCTRA work area.



<signal_layer_id>

The name of a signal layer defined in the design file.

<system_layer>

The name of a system layer in SPECCTRA. Each system layer provides a visual feature such as guides or component labels.

Use the **vset** command to minimize screen repainting when changing layer visibility in the SPECCTRA work area. A <*signal_layer_id*> is the name of a signal layer defined in the design file and a <*system_layer*> is the name of a system layer in SPECCTRA. Each system layer provides a visual feature such as guides or component labels.

You can specify one or more keywords separated by a space in a single **vset** command. After using one or more **vset** commands to turn on or turn off signal layers and system layers, use the repaint command to update the display in the work area.

The following keywords represent the system layers that you can view for routing and placement:

component_labels
error
grid
keepout

origin
pin
power
power_pins
region
site
unroute
via
via_grid
wire

If you use **vset** to display component labels or routing guides (unroutes), you can use the **show** command to control what kind of labels or guides you want SPECCTRA to display. You can:

- Use the show component_labels command to display component names (reference designators), pin IDs, component names with pin IDs, component cluster names, image names, logical or physical part IDs, or virtual pin IDs.
- Use the show unroutes command to display guides for all unrouted nets, guides for unrouted signal nets connected to placed or selected components, or guides for unrouted power nets.

At the beginning of a session, the default for **vset component_labels** is **off** and the default for **vset unroutes** is **on**.

Notes

Additional placement keywords that represent system layers that you can view are:

place_error
place_back
place_front
place_grid

You can control the density, histogram, and force_vector displays by using these keywords with the **vset** command, or by choosing commands from the Autoplace menu.

You can also control the viewing options in the Layers panel. The optional **system** keyword, in the **vset** command that appears in the output window when you use the Layers panel to turn on or turn off a system layer, is used in case a signal layer has the same name (<*signal_layer_id*>) as the system layer.

See also

view (placement)
view grid

Command examples

vset L3
repaint

vset wire off
repaint

vset L1 L2 on
vset pin on
repaint

<system_layer>

component_labels

Displays or hides labels that identify components, pins, virtual pins, images, logical parts, physical parts, virtual pins, or component clusters, depending on the which label **type** you choose with the **show component_labels** command.

error

Displays or hides routing rule conflict and violation symbols for all visible signal layers.

grid

Displays or hides the wire grid if defined. You can use the **view grid** command to control whether the wire grid is displayed as dots or lines.

keepout

Displays or hides keepout areas on all visible signal layers.

origin

Displays or hides component origins (when component outlines are visible) on all visible signal layers.

pin

Displays or hides component pins (when component outlines are visible) on all visible signal layers.

power

Displays or hides guides (flight lines) that show unrouted power net connections.

power_pins

Displays or hides power (P) and ground (G) labels on power pins when component outlines are visible.

region

Displays or hides region boundaries on all visible signal layers.

site

Displays or hides the placement sites of selected images.

unroute

Displays or hides guides (flight lines) that show unrouted pin-to-pin connections.

via

Displays or hides vias for all visible signal layers.

via_grid

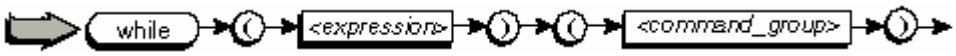
Displays or hides the via grid if defined.

wire

Displays or hides routed wires for all visible signal layers.

while

The **while** command evaluates <*expression*> to determine whether commands within the loop are run. The commands in the loop are repeatedly run until <*expression*> is zero.



If <*expression*> evaluates to a non-zero value, commands within <*command_group*> are run. The expression is evaluated again and the cycle is repeated. When <*expression*> evaluates to zero (false), the loop terminates.

Be careful to avoid endless loops. Control the loop with a counter, which is incremented or decremented within the loop and checked at the start of each pass through the loop.

If you are running a do file and you think the autorouter is in an endless **while** loop, you can type **stop** to terminate the do file. This is the same as clicking the Pause button and then clicking the Stop button in the GUI.

The internal autorouter variables that can be used with this command are defined under <*system_variable*> in the *Design Language Reference*.

Command examples

```
route 25
setexpr count (5)
while (count > 0 && conflict_wire > 10)
    (route 10 16
    clean 2
    setexpr count (count -1)
    )
clean 2
write wires wires.w
```

wildcard

The **wildcard** command defines an alternative character to replace the asterisk (*) character for use as a wildcard when you run commands.

You can use this command when the asterisk (*) character occurs in your design file as part of a string name such as net name, component ID, image name, layer name, or padstack name.

Replace <*character*> with the symbol you want to use instead of asterisk (*). If you specify a character that is already used in your design, a message popup dialog box displays with a list of characters you can use.

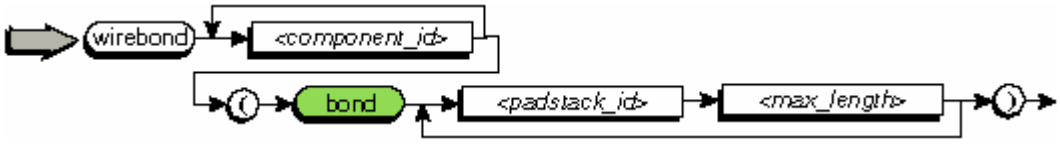
Avoid using alpha and numeric characters, since they are commonly used in a design. In addition, do not use parenthesis, and do not use the quote character defined by **string_quote** in your design file. The default quote character is the apostrophe (').

Command examples

wildcard $
wildcard %

wirebond

The **wirebond** command places bond sites and routes discrete wires from each site to the pads of a chip mounted on the PCB.



bond

Places bond sites. You must

- Identify the bond site padstack name (<*padstack_id*>).
- Specify the maximum distance between the component pad and the bond site (<*max_length*>). The max length must be a positive real number.

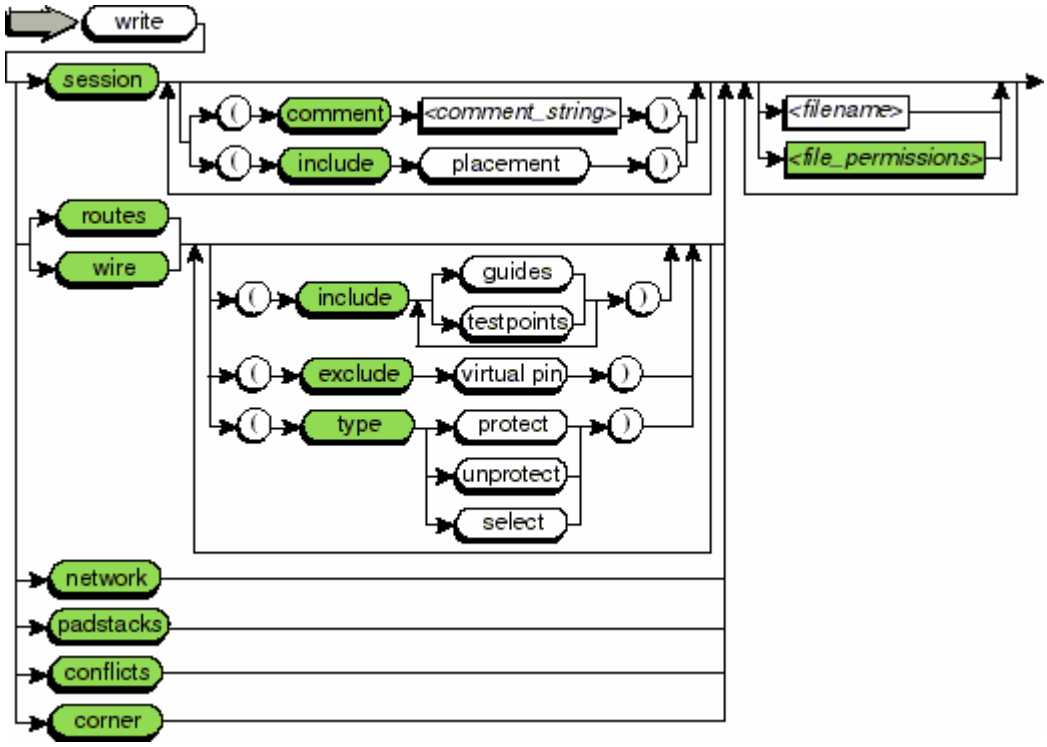
Use the **wirebond** command to automatically route the bond sites of a chip (<*component_id*>) mounted on your PCB. During the wirebond operation, SPECCTRA automatically places bond sites based on your selection of padstacks and specified maximum length rule. SPECCTRA completes the interconnection required by the netlist.

Command examples

wirebond U4 (bond P70 100 P55 150)
wirebond U1 (bond site 3 .175 site 4 .175 site 5 .275 site 6 .275)

write

The **write** command saves current design data in a text file.



session

Creates a text file that contains the design filename, a history of previous session files, component placement data, floor plan data, and route data. You can

- Use the **comment** option to add documentation information to your session file at the end of the history section.
- Use the **include** option to include placement information even though you have not changed your component placements.

The default filename is design.ses.

comment

Adds documentation information (<*comment_string*>) to your session file at the end of the history section.

include

Includes placement information even though component placement has not changed

when you use the **placement** keyword.

routes

Creates a text file that contains data for all routed wires and vias, plus additional information for translating the route data back to the host layout system. You can use the **include** option to include guideand test point information in the routes file. Use the **exclude** option if you do not want to save virtual pin information in the routes file. You can use the **type** option to include protected, unprotected, and selected wires. to save multiple shape co-linear wire paths as single shape straight wires.

The default filename is design.rte.

wire

Creates a text file that contains data for all routed wires and vias. You can use the **include** option to include guideand test point information in the wire file. Use the **exclude** option if you do not want to save virtual pin information in the wire file. You can use the **type** option to include protected, unprotected, and selected wires to save multiple shape co-linear wire paths as single shape straight wires.

The default filename is design.w.

include

Use this option to include guide and test point information in the wire or routes file. The keywords are:

guides, which adds guide information to the wire or routes file so that the host system can determine the topology used in SPECCTRA for unrouted connections.

testpoints, which adds the <*test_points_descriptor*> section at the end of the wire or routes file. See the *Design Language Reference* for information about the <*test_points_descriptor*>.

exclude

Use this option if you do not want to include virtual pin information in the wire or routes file.

type

Use this option to include data about wires that are protected (**protect**), unprotected (**unprotect**), or selected (**select**) in the wire or routes file.

network

Creates a text file that contains the network supplied in the design file.

The default filename is design.net.

padstacks

Creates a text file that contains images supplied in the design file.

The default filename is design.pad.

conflicts

Creates a text file that contains a list of crossover (cross) and clearance (near) conflicts.

The default filename is design.cnf.

corner

Creates a text file that contains a list of all corners and arcs in the routing. Corners listed are 90 and 135 degrees specifically, and all other angles. Arcs are also listed when round corners are created (requires the appropriate_license).

The default filename is design.crn.

Use the **write** command to save specific routing information in a file that is similar in format to the design file. You can

- Use **session** to save routing information in a session file. If you performed placement in the same session file, you can also save placement information.
- Use **routes** to save routing information in a routes file. The routes file also contains information for translating the route data back to the layout system.
- Use **wire** to save routing information in a wire file.
- Use **network**, **padstacks**, **conflicts**, and **corner** to save routing information in a file. You can extract this information by using another software program.

If you do not specify a filename, SPECCTRA supplies a default filename and saves the file in the design directory. You must specify a filename to save the file in a different directory. See file naming conventions for further details.

On a UNIX system, you can use the <*file_permissions*> option to set read and write permissions on the file.

When you save a session file, you can use the **comment** option to add a comment to the file for documentation purposes. The <*comment_string*> is entered in the file at the end of the history section.

By default, SPECCTRA includes placement information in a session file only when you have performed placement operations during the session. You can use the **include placement** option if you want to include placement information even though you have not changed your component placements.

To include guides and test point information in the routes file or wire file, you can use the **include** option. To exclude virtual pin information in the routes file or wire file, click the **exclude** option. To include wires that are protected, unprotected, or selected in the routes file or wire file, you can use the **type** option.

You can also include guide information in the routes file, by inserting (routes_include guides) in the parser section of the design file. See the *Design Language Reference* for more information. Do so only if your translator can parse the guide information in the routes file.

Notes

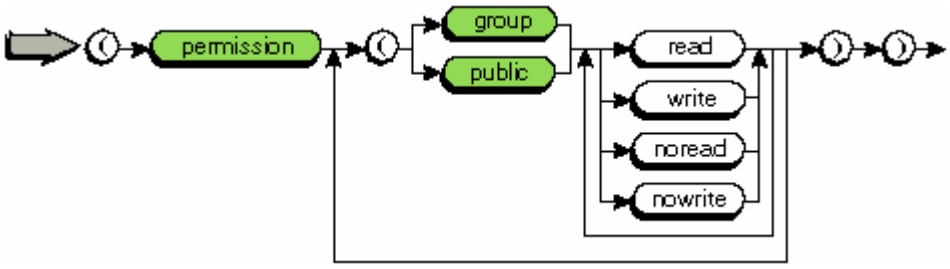
The session file does not include any definitions or design rules you set or changed during the session. If you want to save the rules and definitions you applied during the session, generate a did file, edit it using a text editor, and run it as a do file when you restart the session.

Command examples

write wires final.w
write routes (include testpoints)
write routes (type protect)
write session (comment new bypass caps added)
write session (permission (group read write) (public read nowrite))
write network

<file_permissions>

The <*file_permissions*> option controls read and write access for files you save with the write command on UNIX systems.



permission

Controls whether read and write permissions are set to group access (**group**) or public access (**public**) for files you create with the **write** command.

group

Sets group access to files and directories. The permissions are **read**, **write**, **noread**, **nowrite**.

public

Sets public access to files and directories. The permissions are **read**, **write**, **noread**, **nowrite**.

Use <*file_permissions*> to control read and write access for files you save with the **write** command on UNIX systems.

You can set both **group** or **public** read and write permissions on files that you own.

If you do not specify <*file_permissions*>, your default permissions are used when you save a new file with the **write** command. If you overwrite a file, the permissions are unchanged.

Note

The owner of a file always has read and write access.

Session file

When you achieve satisfactory placement and routing results, save the information in a session file before you exit SPECCTRA. You can use this file to

- Restart the session at a later time
- Translate the design back to your layout system.

A session file contains the design filename, a list of previous session files, a list of other files generated during the session, and the current design status (which can include placement, floor plan, and routing information, depending on the **write** command options you use and the tasks you performed during the session). For placement, the swap list updates your netlist with the new net-to-pin assignments and is created only if you performed a swap operation during the session. Floor plan information consists of cluster and room definitions.

If you use a session file to restart a session, SPECCTRA reads the session file, loads the design file identified in the session file, loads the placement, floor plan, and routing information contained in the session file, and applies any swap data contained in the session file.

Routes file

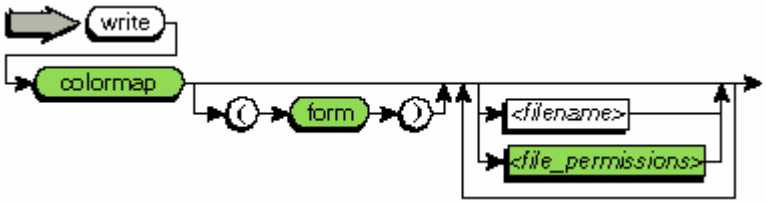
The routing information in a routes file includes wires and vias. It also includes information for translating route data back to the layout system. The routes file includes guides and test point information if you use the **include** option with the **guides** or **testpoints** keyword. It includes wires that are protected, unprotected, or selected if you use the **type** option with the **protect**, **unprotect**, and **select** keyword. You can use the read routes command to reapply this data, except for **type select** data.

Wire file

The routing information in a wire file includes wires and vias data. The wire file includes guides and test point information if you use the **include** option with the **guides** or **testpoints** keyword. It includes wires that are protected, unprotected, or selected if you use the **type** option with the **protect**, **unprotect**, and **select** keyword. You can use the read wire command to reapply this data, except for **type select** data.

write colormap

The **write colormap** command saves current color map information in a text file.



colormap

Creates a text file that contains data that defines colors used in the SPECCTRA work area and assigns colors and fill patterns to design objects and graphical features.

You can use the **form** option to create a color map file that uses the current color pattern settings from the Color Palette dialog box instead of the settings currently in the SPECCTRA work area.

The default filename is color.std.

form

Specifies that the current color pattern settings from the color palette rather than the settings currently in the SPECCTRA are used when creating a color map file.

Use the **write colormap** command to save color map information in a colormap file.

If you do not specify a filename, SPECCTRA supplies a default filename and saves the file in the design directory. You must specify a filename to save the file in a different directory. See file naming conventions for further details.

On a UNIX system, you can use the <*file_permissions*> option to set read and write permissions on the file.

The colormap file contains data that defines colors used in the SPECCTRA work area and assigns colors and fill patterns to design objects and graphical features.

To create a color map file that uses the current color pattern settings from the Color Palette dialog box instead of the settings currently in the SPECCTRA work area, you can use the **form** option.

If you do not provide a color map file, SPECCTRA uses colors and patterns defined and mapped in the design file, or uses internal defaults.

You can use the read colormap to reapply the data in the color map file.

Note

You can use write environment to save the current colormap and key definitions in your home directory.

Command examples

write colormap
write colormap (form)
write colormap color3.std (permission public nowrite)

write environment

The **write environment** command saves the current SPECCTRA color map, key definitions, or both in your .cct directory.

environment

Creates text files, in your .cct directory, that contain environment data from the current session. You can save a color map file (**colors**), a key definitions file (**keys**), or both.

Use the **write environment** command to save your current color map and key definitions in text files for use when you start the next session. You can

- Use the **colors** option to save just the color map.
- Use the **keys** option to save just the key definitions.

Both the color map and the key definitions are saved by default if you use this command without either of these options.

On a UNIX system, you can use *<file_permissions>* options to set read and write permissions on the file.

SPECCTRA saves the color map in a file named colors and the key definitions in a file named keys. The files are located in a directory named .cct under your home directory. If the .cct directory does not exist, SPECCTRA creates it for you.

The location of the .cct directory on Windows systems depends on how certain environment variables are set.

- On Windows NT systems, the .cct directory is located under the directory defined by the %homedrive% and %homepath% environment variables. For example

  HOMEDRIVE=D
  HOMEPATH=\users\myname

  SPECCTRA saves the colors and keys files in D:\users\myname\.cct.

- On Windows 95 systems, the .cct directory is located under the Windows directory (declared in the WINDIR environment variable). For example

  WINDIR=c:\win95

  SPECCTRA saves the colors and keys files in C:\win95\.cct.

Notes

You can save the color map in a different file or directory using the write colormap command, and you can load a color map saved in a different file or directory using the read colormap command.

You can save key definitions in a different file or directory using the write keys command. The key definitions are saved as a series of defkey commands.

When you start SPECCTRA, it looks in your .cct directory for these files and, if either or both of them exist, loads them before processing any do files that you specified. Colors or fill patterns defined or assigned in the design file override those definitions or assignments in the .cct directory colors file.

Use the -noinit startup switch if you want to prevent SPECCTRA from loading the colors and keys files.

You can use the -c startup switch (or the Color Mapping File option in the Startup dialog box) to specify a different color map file than the colors file in the .cct directory. Colors or fill patterns defined or assigned in the file you specify with -c override those definitions and assignments in the design file.

If you do not use -c, and either you use -noinit or the .cct directory does not contain a colors file, SPECCTRA looks for a file named color.std in the current directory. If this file does not exist, SPECCTRA uses color and fill pattern definitions and assignments in the design file, or internal defaults.

You can use the -do startup switch (or the Do File option in the Startup dialog box) to load key definitions from a different file when you start a session, or using the do command to load key definitions from a file any time during a session. Keys defined in a do file override those key definitions in the .cct directory keys file.

See chapter 2 in the *SPECCTRA User Guide* for details about using startup options.

Command examples

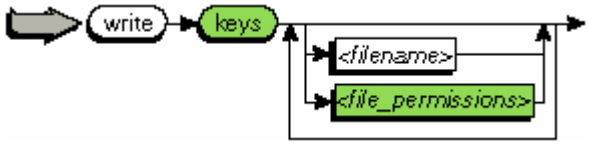
write environment

write environment (colors)

write environment (permission (group read nowrite)) (keys)

write keys

The write keys command saves key definitions in a text file.



keys

Saves key definitions in a text file that consists of a series of **defkey** commands.

The default filename is defkey.std.

Use <*file_permissions*> to control read and write access for files you save with the **write** command on UNIX systems.

You can set both **group** or **public** read and write permissions on files that you own.

If you do not specify <*file_permissions*>, your default permissions are used when you save a new file with the **write** command. If you overwrite a file, the permissions are unchanged.

Note

The owner of a file always has read and write access.

Use the **write keys** command to save key definitions in a text file. The text file consists of a series of **defkey** commands. It is a do file that you can use to define the same keys during a future SPECCTRA session.

If you do not specify a filename, SPECCTRA supplies a default filename and saves the file in the design directory. You must specify a filename to save the file in a different directory. See file naming conventions for further details.

On a UNIX system, you can use the <*file_permissions*> option to set read and write permissions on the file.

Note

Some key definitions that you save in the text file might not be definable if you try to use them on another system.

You can use write environment to save the current colormap and key definitions in your home directory.

Command examples

write keys
write keys (permission public nowrite)