cadence®

Virtuoso Space-based Router Constraint Reference

Product Version IC23.1
June 2023

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

11
Mixed-Signal Routing Constraints 523

15
Routing Constraints

1

Manage Constraints

This topic describes the constraint components for Virtuoso Space-based Router.

Process rules, or constraints, specify conditions that must be met in order for a design to be correct. While many constraints are set by foundry and technology limitations, the Virtuoso space-based router also lets you define your own constraints and apply them to your designs.

There are three constraint components:

- Constraint Objects specify a particular type of constraint and provide a value for a single application of the constraint.
- Constraint Parameters specify additional qualifiers for applying a constraint.
- Constraint Groups and Multispecs let you apply a collection of constraints to specific objects in the design and the technology databases.

Each of these components has predefined or implicit constructs and a user customizable aspect.

Values for constraints and constraint parameters are described in Constraint Value Types.

In addition to applying constraints to individual objects (nets, routes, terms, and area boundaries), you can also group objects and apply constraints to object groups.

Related Topics

Constraint Objects

Constraint Parameters

Constraint Objects

Constraint Group Types

Tcl Constraint Commands

Constraint Value Types

The following constraint value types can be assigned to constraints and constraint parameters:

Value Type	Description
`AreaValue`	Area in user units^2
`BoolValue`	Boolean
`IntValue`	Integer value used for counts and other *true* integers
`FltValue`	Floating-point number
`DblValue`	Double-precision number
`LayerValue`	Layer name
`StringValue`	String value, such as the name of a constraint group
`StringAsIntValue`	String value that is converted to and stored as an integer value
`Value`	Single value in user units, such as a distance or length
`ViaDefValue`	Via definition
`RangeValue`	Range of values.
`DualValue`	Two values in user units
`DualValueTbl`	Table of `DualValue`
`OneDTblValue`	1-D table with `Value` header and `Value` value
`Int1DTblValue`	1-D table with `Value` header and `IntValue` value
`Dbl1DTblValue`	1-D table with `Value` header and `DblValue` value
`LngDbl1DTblValue`	1-D table with `AreaValue` header and `DblValue` value
`OneDDualValueTbl`	1-D table of `DualValue` in user units, indexed by width
`OneDDualArrayTblValue`	Table of dual dimension arrays, in user units, indexed by width. Each row contains a width, the number of extension pairs associated with the width, and the extension pairs.
`Dual1DTblValue`	Two 1-D tables of `Value`
`Flt1DTblValue`	1-D table with `Value` header and `FltValue` value
`FltHeader1DTblValue`	1-D table with `FltValue` header and value in user units

Value Type	Description
`RangeArray1DTblValue`	1-D table of `RangeArrayValue` indexed by width.
`RangeArray2DTblValue`	2-D table of `RangeArrayValue` indexed by width.
`TwoDTblValue`	2-D table of `Value`
`Int2DTblValue`	2-D table of `IntValue`
`Dbl2DTblValue`	2-D table of `DblValue`
`LayerArrayValue`	List of layer names
`RangeArrayValue`	List of `RangeValue`.
`StringArrayValue`	List of string values
`ValueArrayValue`	List of values in user units
`ViaDefArrayValue`	List of via definitions
`CellPathListValue`	List of cellviews
`LayerDualArrayTblValue`	Minimum spacing/minimum parallel length pairs by layer

Related Topics

Range Values

Table Values in Constraints

Table Values in Constraints

Tables are used for constraints whose values are dependent on one, two, or three variables.

1-D tables are represented by a list of ordered pairs. Each pair includes a header value and a constraint value. The header entries, also referred to as *row* values, typically represent width. To find a constraint value in a 1-D table, Space-based Router and Chip Optimizer performs a table lookup with your row *key* value or row index. If the row key value does not match a header entry, Space-based Router and Chip Optimizer extrapolates the value if the key lies outside the range of header values or interpolates the value if the key lies between a pair of header values in the table.

For example, a 1-D table list of `{4 2.2 8 3.3 16 4.4}` represents three pairs of values:

Index	Header	Constraint Value

0	4	2.2
1	8	3.3
2	16	4.4

2-D tables are represented by a list of ordered sets and have a related list of header *column* values that typically represent parallel run lengths. Each ordered set includes a header (row) value followed by one constraint value for each column value. To find a constraint value in a 2-D table, Space-based Router and Chip Optimizer performs a table lookup with your row and column key values or indexes. If key values do not match header entries, Space-based Router and Chip Optimizer extrapolates the value if the key lies outside the range of header values or interpolates if the key lies between a pair of header values in the table.

For example, a 2-D table list of `{0 0.18 0.18 0.3 0.18 .22 10.05 0.6 0.6}` with a column list of `{0 2}` represents the following:

Indexes	-	0	1
-	Header	0	2
0	0	.18	.18
1	0.3	.18	.22
2	10.05	0.6	0.6

Related Topics

Range Values

Constraint Value Types

Interpolation and Extrapolation Techniques

Interpolation and Extrapolation Techniques

When using 1-D and 2-D table values, if the row or column key value does not match a header value, then Virtuoso Space-based Router extrapolates the value if the key lies outside the range of header values or interpolates if the key lies between a pair of header values in the table. You can specify the interpolation and extrapolation technique to use for rows and for columns.

Interpolation Techniques

Interpolation techniques are used when the key value falls between two header values.

- `snap_down`: Returns the constraint value for the next lower header value.
- `snap_up`: Returns the constraint value for the next higher header value.
- `linear`: Returns a constraint value that is between the values for the next lower and the next higher values, in proportion to where the key is in that range.
- `snap_down_inclusive`: Returns the constraint value for the next lower header value, including when the key matches the higher header value.
- `snap_up_inclusive`: Returns the constraint value for the next higher header value, including when the key matches the lower header value.

In Virtuoso ASCII technology file data, the value of the table index is '>=' for most tables which corresponds to `snap_down` interpolation. For all LEF spacingTables and Virtuoso ASCII technology file 2-D `minSpacing` tables, the value of the table index is '>' which corresponds to `snap_down_inclusive` interpolation. Due to this discrepancy, you will see that many of the constraint definition examples in this manual will show slightly different row and column header values for LEF and Virtuoso. The Tcl description will follow one or the other, depending on the row/column interpolation setting.

Extrapolation Techniques

When setting a 1-D or 2-D table value, you can specify the extrapolation technique to use when the key value is lower than the lowest header value and another extrapolation technique for use when the key value is higher than the highest header value.

- `snap_up`: If the key is smaller than the lowest header value, then return the constraint value for the lowest header value.
- `snap_down`: If the key is greater than the highest header value, then return the constraint value for the highest header value.
- `linear`: Returns a constraint value that is extrapolated from the two points at the end (either lower or upper) to the key value.

Typically, `{snap_up snap_down}` is used, meaning snap up to the lowest value on the low end and snap down to the highest value on the high end.

Related Topics

Range Values

Table Values in Constraints

Range Values

Range values used for constraints are as follows.

- **RangeValue**

  Specifies a range in the following formats:

Format	Range is
"[*f_x f_y*]"	>= *f_x* and <= *f_y*
"(*f_x f_y*)"	> *f_x* and < *f_y*
"[*f_x f_y*)"	>= *f_x* and < *f_y*
"(*f_x f_y*]"	> *f_x* and <= *f_y*
"< *f_x*"	less than *f_x*
"<= *f_x*"	less than or equal to *f_x*
"> *f_x*"	greater than *f_x*
">= *f_x*"	greater than or equal to *f_x*
f_x	equal to *f_x*

- **RangeArrayValue**

  Specifies a list of `RangeValue` values.

  For example,

  ```
  { 0.1 0.2 ">= 0.35" }
  ```

  specifies that values of 0.1, 0.2, and greater than or equal to 0.35 are permitted.

- **RangeArray1DTblValue**

  Specifies a table of RangeArrayValue (as shown in Table 1-1) indexed by width. Each row contains a width, the count of RangeArrayValue for the given width, and the RangeArrayValue list.

{ *f_width1 i_count* {*s_range1*$_1$ … *s_range1*$_{count}$} \
…
f_widthN i_count {*s_rangeN*$_1$ … *s_rangeN*$_{count}$} **}**

Tables must be specified in ascending index value order. Each table entry is read as greater or equal to the index value, but less than the next index value. For the following example,

```
-RangeArray1DTblValue {0.3 2 0.3 0.4 \
                       0.4 3 0.3 0.4 0.5 \
                       0.5 2 0.4 ">= 0.5"}
```

- When the index value is greater than or equal to 0.3, but less than 0.4, the constraint value may be 0.3 or 0.4.
- When the index value is greater than or equal to 0.4, but less than 0.5, the constraint value may be 0.3, 0.4, or 0.5.
- When the index value is greater than or equal to 0.5, the constraint value may be 0.4 or greater than or equal to 0.5.

The first value in the table that matches a query of the lookup table is used. In this example, for an index value of 0.41, the constraint value may be 0.3, 0.4, or 0.5.

■ RangeArray2DTblValue

Specifies a table of RangeArrayValue (as shown in Table 1-1) indexed by width and parallel run length. We first give –TblCols values for each column and then RangeArray2DTblValue where each ordered set includes a header (row) value followed by one RangeArray for each column value. Range Array is given within Curly brackets {}.

In the following example, –TblCols specifies 3 values which means there are 3 columns in the –RangeArray2DTblValue, each set has one value for row header and 3 separate Range arrays. Blank RangeArrays can also be given.

```
set_layer_constraint -constraint oaAllowedSpacingRange -layer Metal1 \
-TblCols {0.0 0.2 0.4}\
```

Tables must be specified in ascending index value order. Each table entry is read as greater or equal to the index value, but less than the next index value. For the following example,

```
-RangeArray2DTblValue \
{0.0 {"(0.1 0.2)" "[0.3 0.4]" ">=0.5"} {} {} \
0.2 {} {"(0.1 0.3)" ">=0.5"} {"(0.1 0.4]" ">=0.6"} \
0.4 {} {"(0.1 0.4]" ">=0.6"} {"(0.1 0.5]" ">=0.7"}}
```

- For (index1, index2) = (0.0, 0.0), the constraint value is (0.1, 0.2), or (0.3, 0.4), or greater than or equal to 0.5.
- For (index1, index2) = (0.2, 0.2), the constraint value is (0.1, 0.3), or greater than or equal to 0.5.

- For (index1, index2) = (0.4, 0.4), the constraint value is (0.1, 0.5), or greater than or equal to 0.7.
- For (index1, index2) = (0.2, 0.4) and (0.4, 0.2), the constraint value is (0.1, 0.4), or greater than or equal to 0.6.
- For (index1, index2) = (0.0, 0.2), (0.0, 0.4), (0.2, 0.0), (0.4, 0.0), the constraint values are not specified.

Related Topics

Table Values in Constraints

Constraint Value Types

Interpolation and Extrapolation Techniques

Constraint Objects

Virtuoso Space-based Router supports built-in and custom constraints. Many of the Virtuoso Space-based Router built-in constraints are also OpenAccess built-ins because they are defined by OpenAccess. Virtuoso Space-based Router built-in constraints that are not defined by OpenAccess are OpenAccess *custom* constraints. Virtuoso Space-based Router built-in constraints store values derived from other constraints or are used to control a Virtuoso Space-based Router behavior. Virtuoso Space-based Router custom constraints must be defined by the user and are typically used to save constraints for other tools using the same database.

Constraints can be one of the following types:

- *Simple constraints* have no specific layer affiliations.
- *Layer constraints* are associated with a specific layer.
- *Layer pair constraints* are associated with two layers.
- *Layer array constraints* are associated with a layer array of three or more layers.

For the names of built-in constraints, grouped by category, see Constraint Definitions.

Some constraints can be associated with a specific layer or layers, and also with specific layer purpose pairs. For more information on these constraints, see Supported Layer Purpose Pair Constraints.

For information on voltage-specific rules and how to use them, see Voltage-Dependent Rule Support.

Custom Constraints Definition

If you want to set a constraint that is not a built-in, you must first define the constraint. Use the define_constraint command to set the name of the constraint, the constraint type, and its allowed value type.

After you define the constraint, use set_constraint, set_layer_constraint, set_layerpair_constraint, or set_layerarray_constraint to set the value based on the constraint type.

Constraints Setting

When setting constraint values, in addition to the name of the constraint, the value, and the constraint group to apply it to, you can specify the following:

- Whether the constraint is *hard* or *soft*

  A hard constraint must be obeyed. A soft constraint should be obeyed, if possible.

- The antenna model (if applicable)

You can specify additional qualifiers for some constraints by setting constraint parameters. You must set the constraint's parameters immediately prior to setting the constraint value.

To set constraints, use the command appropriate for the constraint type:

- set_constraint sets simple constraints for the design.
- set_layer_constraint sets constraints that apply to individual layers.
- set_layerpair_constraint sets constraints that apply to two layers.
- set_layerarray_constraint sets constraints that apply to three or more layers.

Constraint Values Retrieval

Use the following commands to get constraint values:

- get_constraint returns the value for a simple constraint.
- get_layer_constraint returns the value for a single-layer constraint.

- get_layerpair_constraint returns the value for a double-layer constraint.
- get_layerarray_constraint returns the value for a layer array constraint.
- dump_ctu_constraints outputs all the constraint values from the current Space-based Router and Chip Optimizer database using the Tcl commands for setting the values. Derived and local constraints are included with built-in OA constraints. Options allow you to output to a file, specify the OA namespace, and limit the scope of the data output to a specific constraint or constraint groups.
- dump_oa_constraints outputs the constraint values from the current OA database using the Tcl commands for setting the values. Can optionally be output to a file.

Constraints Removal

Use the undefine_constraint command to remove a constraint from all constraint groups and remove its definition.

To selectively remove a constraint value, use the appropriate command for the constraint type:

- unset_constraint removes simple constraint values.
- unset_layer_constraint removes single-layer constraint values.
- unset_layerpair_constraint removes double-layer constraint values.
- unset_layerarray_constraint removes layer array constraint values.

Related Topics

Layer-Purpose Pair Constraints

Voltage-Dependent Rule Support

Constraint Definitions

Constraint Parameters

Constraint Parameters

You can set constraint parameters to specify additional qualifiers for applying the constraint. For example, the `distance` parameter for the minimum adjacent via constraint indicates that the constraint applies only if the adjacent via is within the specified distance to the object.

Constraint Parameters Setting

Use the set_constraint_parameter command to add a parameter to a constraint or to change the value of an existing constraint parameter.

You must issue one command for each constraint parameter, followed by a set_constraint, set_layer_constraint, set_layerpair_constraint, or set_layerarray_constraint command to associate the parameters with a constraint. If any errors occur with any of the commands in this process, the pending constraint addition or change, and any constraint parameters are lost and the entire command sequence must be redone.

The order of constraint parameters is not significant. If a constraint requires more than one parameter, any parameter can be set first, as long as all are specified before the command is issued to set the constraint.

Related Topics

Constraint Definitions

Constraint Groups and Multispecs

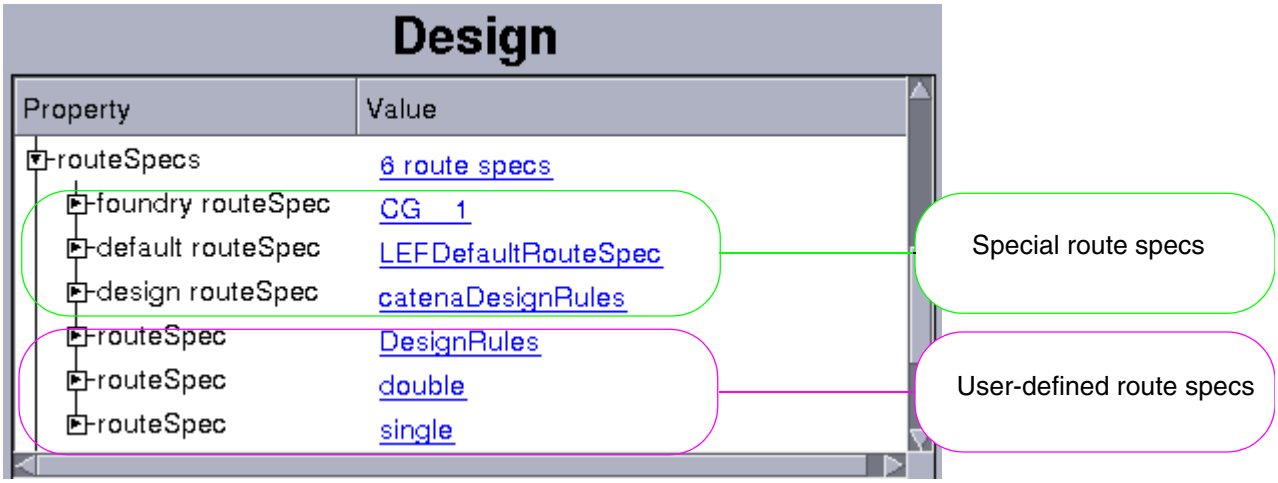
Constraint Groups and Multispecs

In a design, you can have numerous *route specs*, also known as *rule specs* or *constraint groups*. The route specs contain the rules, or constraints, that must be obeyed in order for the design to be correct. Each route spec has a name, and three of the route specs can have special designations:

- The **foundry** route spec comes from the technology library and specifies rules typically given by the foundry or manufacturer.
- The **design** route spec is the default constraint group for the design and includes derived constraints. If no constraint group has been set as the design route spec, then a `catenaDesignRules` constraint group is created for that purpose.
- The **default** route spec includes routing information and is typically called `LEFDefaultRouteSpec` when translated from LEF. The default route spec is also referred to as the *Global net default* route spec.

All other route specs are considered to be user-defined route specs. For a list of route specs defined in a design, use the Properties Browser and view the `routeSpecs` property for the

design. The Browser's display defaults to the design's properties when no objects are selected.



One route spec can be identified as the **override** route spec. When set, the override route spec takes precedence over all other route specs for rule lookups. The override setting is not stored persistently.

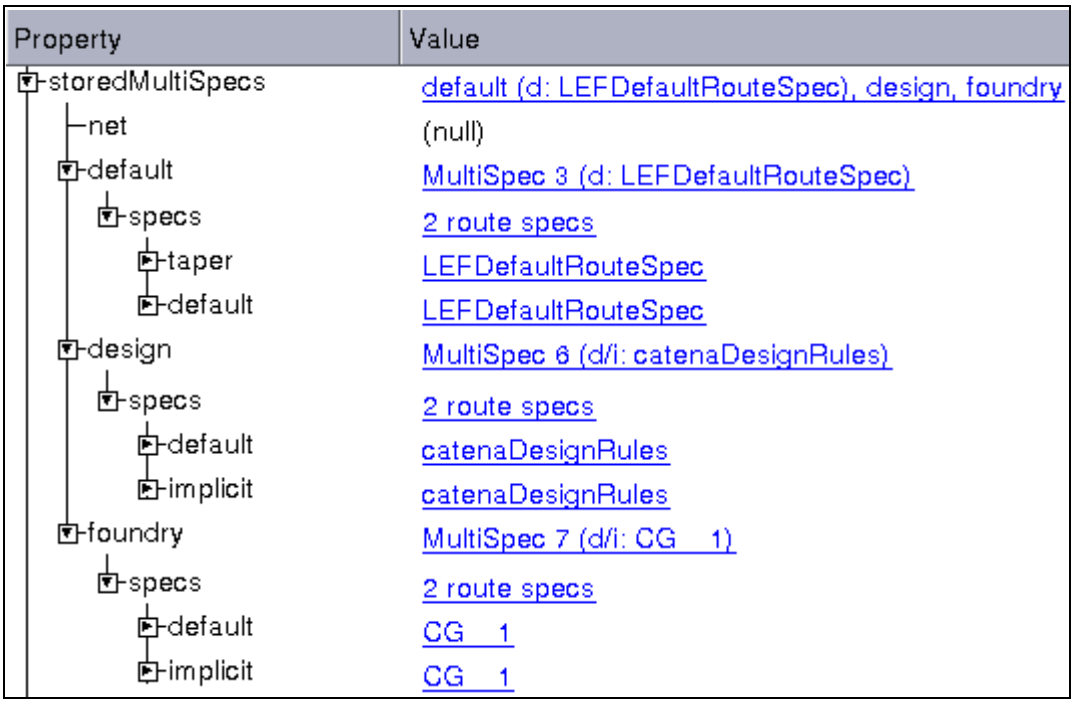
Each design object has an associated *multispec*. The multispec is an array of route specs that have been assigned to the object as specific constraint group types. The constraint group type specifies how constraints in the route spec are used. For example, when a route spec is assigned to the *shield* group for an object, the constraints in that route spec apply only for shielding. If no constraint groups have been assigned to an object, the object has a **NULL multispec**.

In addition to the multispecs assigned to objects, there is also a **default multispec**.

The following figure shows an example of the stored multispecs for a net displayed by the Properties Browser.

- The net's multispec is NULL, meaning that no constraints groups were directly assigned to the net.
- The global net default route spec, `LEFDefaultRouteSpec`, is the default constraint group for the default multispec.
- The global net default route spec, `LEFDefaultRouteSpec`, is also the taper constraint group for the default multispec.

Example of Stored Multispecs for a Net



For the list of multispecs associated with any object, select the object, then view the `storedMultiSpecs` for the object in the Properties Browser.

The value of a constraint for an object is determined by the hierarchy of rule specs and objects.

The constraint group *operator* (precedence, AND, or OR) determines how member constraints apply to an object.

Currently, constraint groups can only be assigned to and removed from nets, routes, terms, and area boundaries.

Related Topics

set_override_rulespec

Constraint Group Types

Applying Constraints to Objects

Constraint Group Types

When assigning constraint groups to objects, you specify how the constraints in the route spec will be used by assigning the group type. The table below lists the constraint group types with their descriptions and the objects that they can apply to.

Constraint Group Type	Description	Objects
`implicit`	Constraints apply to the object itself.	All objects
`default`	Constraints apply to contained objects but not to the container objects. This is the constraint group type for typical default and non-default (for example, wide wires, double spacing) constraint groups.	Technology database objects, design database objects, blocks, modules, nets, routes, groups, fig groups, boundary objects
`foundry`	Constraints are associated with a technology database. A technology database can have only one foundry constraint group.	Technology database objects
`taper`	Constraints apply only for tapering.	Single-bit terminals, instance terminals, pins
`inputTaper`	Constraints apply only to input pin tapers on nets.	Single-bit nets
`outputTaper`	Constraints apply only to output pin tapers on nets.	Single-bit nets
`shield`	Constraints apply only to objects for shielding.	Single-bit net, nets
`transreflexive`	Constraints apply only between members of the group and objects not in the group.	Object groups
`reflexive`	Constraints apply only between members of the group.	Object groups
`interChild`	Constraints apply between objects within subgroups but do not apply between objects within the parent group.	Object groups

Constraint Group Type	Description	Objects
`nearfarend`	Constraints that can override the default or implicit rulespec settings for a percentage of a net, based on the nearFarPercentage constraint setting.	Nets
`userdefined`	Collects all user-named constraint groups that are associated with a database but not associated with a particular object. User-defined constraint groups in older databases are updated to be associated with this constraint definition.	Technology and design database objects

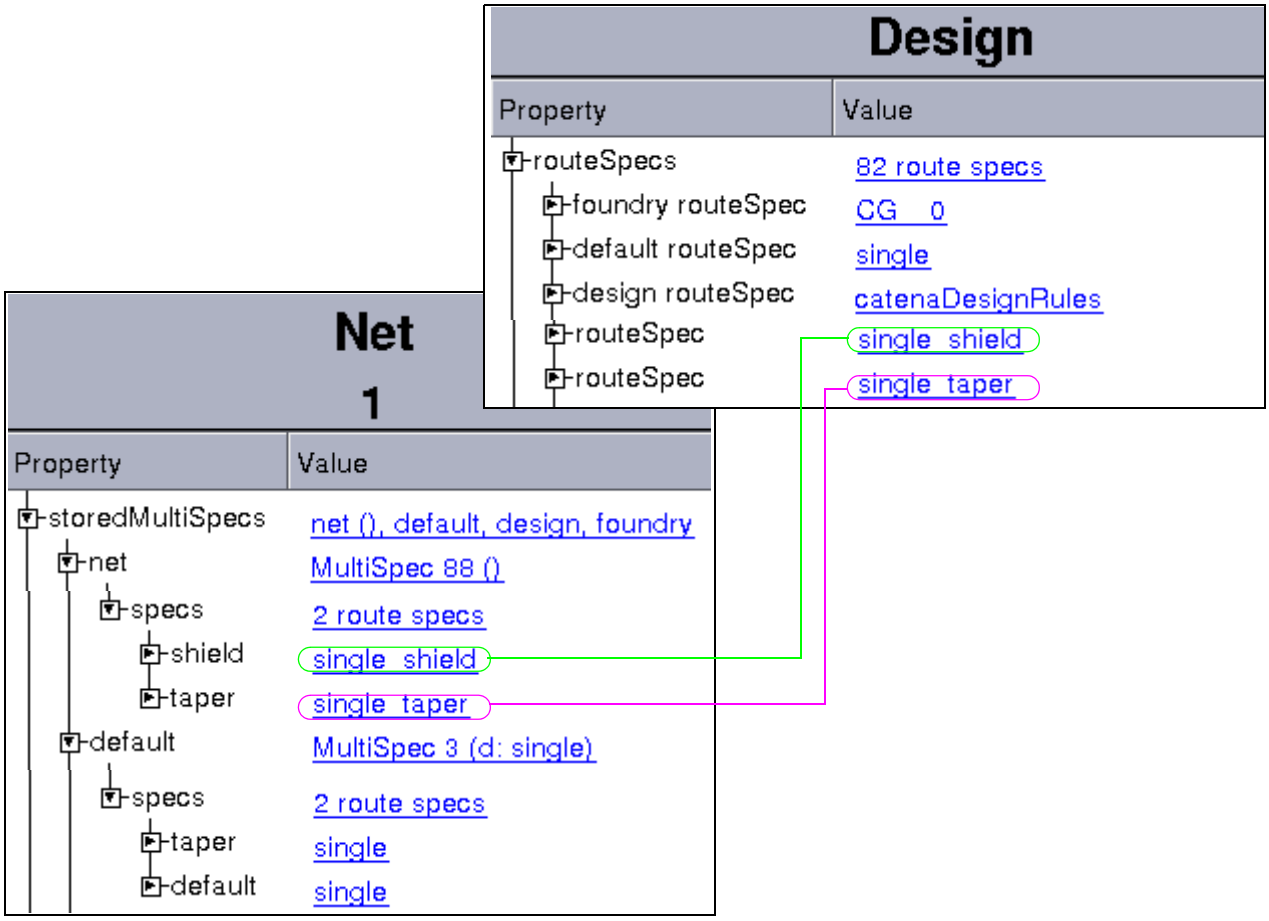
The `implicit` constraint group is a convention used by OpenAccess. The `default` constraint group is used by other tools to represent the rules that should be applied to an object under normal circumstances. If the `default` constraint group exists, it will be used, otherwise, the `implicit` rules will be used.

Use set_constraint_group to assign route specs to objects by constraint group type. For example, consider the following command:

```
set_constraint_group -net 1 -taper single_taper -shield single_shield
```

The `shield` and `taper` constraint groups are assigned to net `1` and can be viewed in the Properties Browser in the `storedMultiSpecs` property for the net as shown in the following figure. The constraints set in the `single_shield` constraint group are used for shielding, while the constraints set in the `single_taper` constraint group are used for tapering on the net.

Example of Assigning Constraints Groups to a Net



TransReflexive, Reflexive, and InterChild Constraint Groups

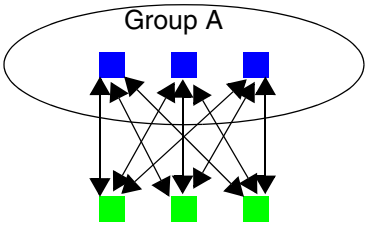
These types of constraint groups are intended to apply to objects within an object group, which is a collection of design objects. These types of constraint groups provide more flexibility and a greater degree of control for applying constraints.

TransReflexive

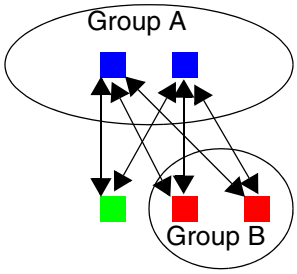
TransReflexive constraints apply between each object in an object group and all objects outside the object group as shown in the following figure. The constraints do not apply between objects in the group, nor do they apply between objects outside the group. If the group contains a sub-group, then the transReflexive constraints apply between the sub-group objects and all objects outside the group, but the constraints do not apply between objects within the sub-groups nor between objects in the sub-groups and other sub-groups, or objects

in the parent group. This type of constraint group is useful for cases such as net pairs and buses. For example, to minimize noise around a net pair, you could specify a larger spacing as a transReflexive constraint for the net pair, which would require a bigger gap between the net pair and surrounding nets.

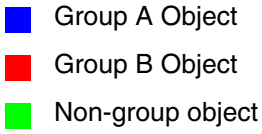
TransReflexive Constraint Group



a) Transreflexive constraints for Group A objects apply between objects in Group A and all objects outside of Group A.



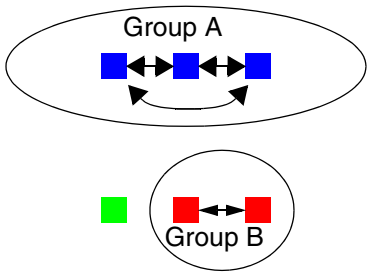
b) Transreflexive constraints for the net pair in Group A apply between each net of the net pair and all other objects outside of Group A, including objects in other groups.

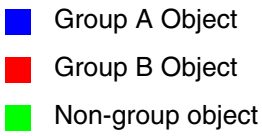


Reflexive

Reflexive constraints apply between the objects in a group as shown in the following figure. This type of constraint group is useful for cases such as net pairs and buses. For example, reflexive constraints can specify the gap between the nets of a net pair that is different from the spacing required between the net pair and other nets.

Reflexive Constraint Group



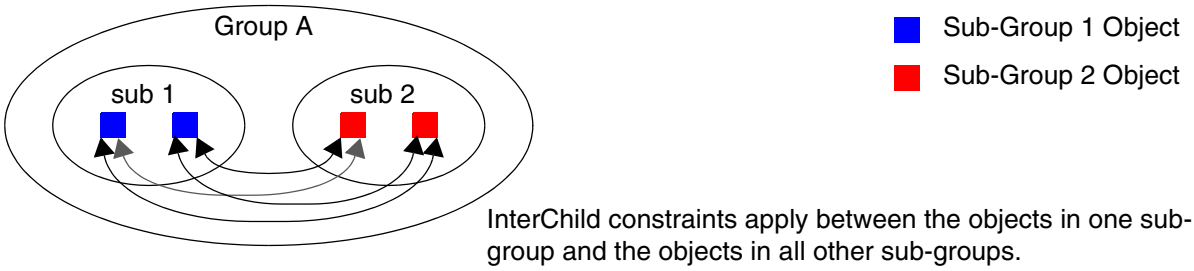


Reflexive constraints for Group A objects apply only between objects within the group and do not apply between the group object and objects outside of the group.

InterChild

InterChild constraints are also referred to as *Group-to-Group* constraints. Constraints in an interChild constraint group apply between the relevant objects in a sub-group and all objects in other sub-groups of the same parent group as shown in the following figure. For example, to route some signal nets close to power rails, you can create a group of the signal nets, a group of power nets, and identify the groups as good neighbors by specifying `crosstalkNeighborIndex` as an interChild constraint for the two groups. If interChild constraints are applied to a group that has no sub-groups, then the constraints are applied as reflexive constraints.

InterChild (Group-to-Group) Constraint Group



InterChild constraints apply between the objects in one sub-group and the objects in all other sub-groups.

Related Topics

Constraint Parameters

Constraint Groups and Multispecs

Applying Constraints to Objects

Simplified Constraint Search Hierarchy

There are two types of hierarchies that determine the value of a constraint for an object: Rule Spec Hierarchy and Object Hierarchy.

Rule Spec Hierarchy

If a constraint is not defined on a certain rule spec, that constraint might be defined by another rule spec in the fallback sequence. The rules system applies the fallback sequence to determine the constraint value to use.

Object Hierarchy

The object hierarchy first looks for the constraint in the object's constraint group. If the constraint is not found, the search extends to the architecture of the object's owners, in hierarchical order.

Virtuoso Space-based Router will search for all constraints in the Override, Design, and Foundry route specs, but in other levels of the hierarchy, only specific constraints are searched for, as shown in the following figure.

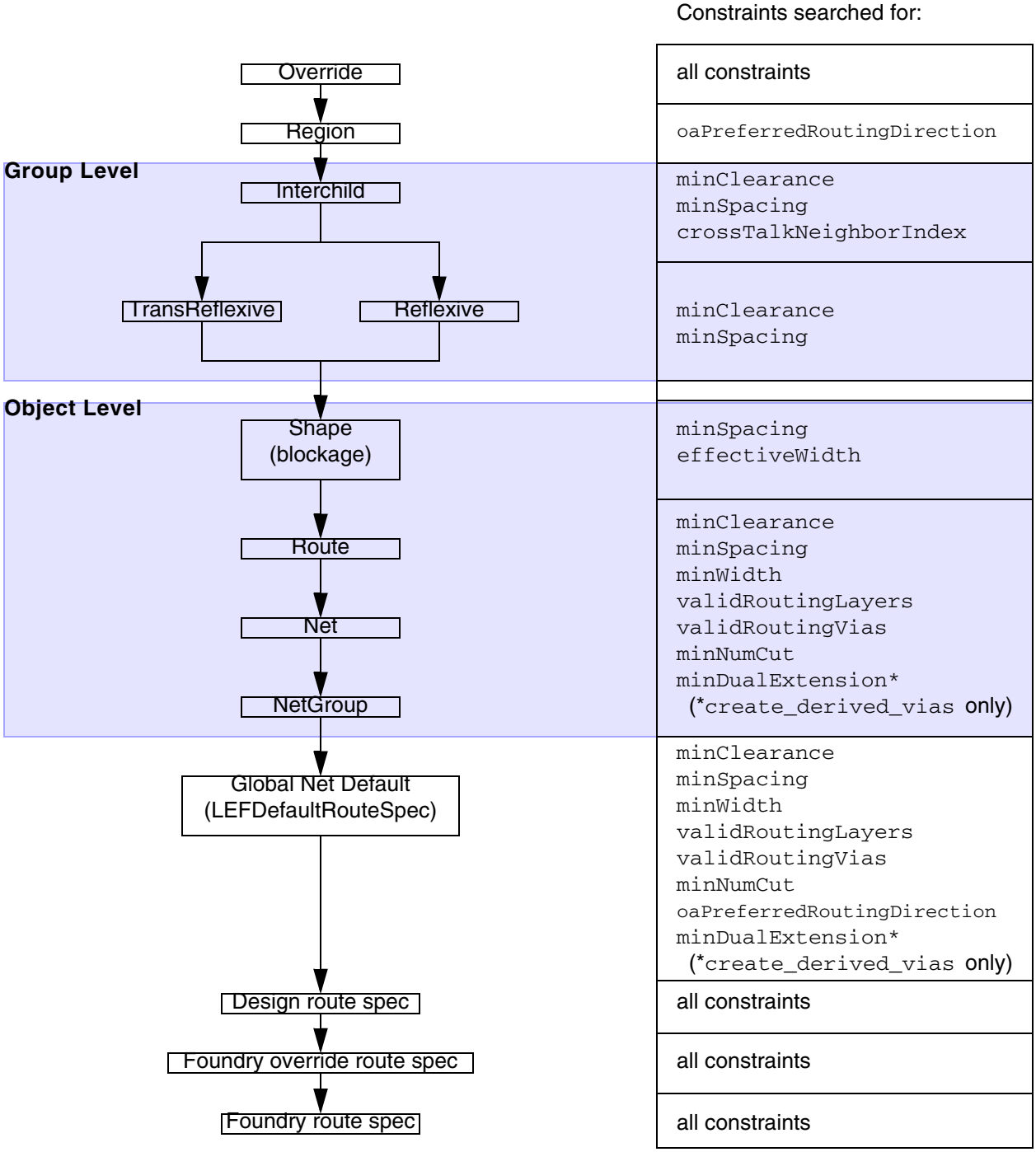
When setting constraints, apply the constraint to the highest level of hierarchy for which it is appropriate.

In addition to deciding the appropriate scope for your constraint in the object hierarchy, you must decide whether your constraint is application-specific. Application-specific constraints influence a particular application's behavior, but might conflict with the requirements of another application.

For example, if you add an application-specific constraint to the `Design` default constraint group that limits the valid routing layers to `Metal3`, `Metal4`, and `Metal5`, your application might be required to route only `Metal3`, `Metal4`, and `Metal5`, but it is not reasonable to restrict all other applications to use only those layers.

Application-specific constraints should *not* be added to the built-in constraint groups, which apply to all applications. Instead, place your application-specific constraints into an application-specific constraint group and leave the application-specific constraint group unattached from any built-in constraint group, and the application can find it by name.

Simplified Constraint Search Hierarchy



Override Rule Spec

At the highest level, if an override rule spec has been set , then all constraint values are taken from that rule spec. If the desired constraint does not exist in the override rule spec, then the search goes to the Region.

Region

The oaPreferredRoutingDirection constraint can only be assigned to an area boundary, or region, which is specified using the create_preferred_direction_region Tcl command. This overrides the global settings for the preferred routing direction in the region. If the desired constraint does not exist for the region, then the search goes to the Group Level.

Group Level

The Group level applies only for constraints that are set in an InterChild, TransReflexive, or Reflexive constraint group for a group, typically a netGroup. InterChild constraints take precedence over the other two types.

Only the minSpacing and crossTalkNeighborIndex constraints are searched for in InterChild constraint groups. For TransReflexive, and Reflexive constraints, the search is restricted to minSpacing.

If there are no groups defined or the constraint is not defined for a group constraint group, the search continues to the Object Level.

Object Level

The Object level applies for the constraints that are assigned to blockage shapes, routes, nets, and netGroups, as shown in Simplified Constraint Search Hierarchy. At this level, the search begins with the object, then to the object's parent, and so on, until the constraint is found. If the constraint is not found, the search goes to the Global Net Default Route Spec, the Design Rules, and, finally, the foundry rules.

To apply a constraint to an object, the constraint is added to a specific constraint group that is assigned to the object as shown in the table below. The function of the constraint group determines how the constraint is used. For example, `minSpacing` on the `shield` group of a net specifies the spacing between the net and its parallel shields, whereas `minSpacing` on the `inputtaper` group for the same net specifies the spacing required when tapering the

net to input pins. The minimum spacing for an unshielded net could be found in the `default` constraint group for the net.

Object Type	Applicable Constraint Group(s)
Blockage shape	`implicit`
Route	`shield`, `default`
InstTerm	`taper`
Term	`taper`
Net	`shield`, `inputTaper`, `outputTaper`, `default`
Net Group	`reflexive`, `transreflexive`, `default`
Group to Group	`interChild`

Global Net Default Route Spec

Only `minSpacing`, `minWidth`, `validRoutingLayers`, `validRoutingVias`, `minNumCut`, and `oaPreferredRoutingDirection` constraints are searched for in the Global net default route spec. In addition, create_derived_vias will use `minDualExtension` from the Global net default route spec if the constraint is set.

If the desired constraint does not exist in or cannot be read from the Global net default route spec, the search goes to the Design Rules.

Design Rules

All constraints that are set in the Design route spec will be read. If the desired constraint does not exist in or cannot be read from the Design route spec, then the search goes to the Foundry Override Rules.

Foundry Override Rules

One constraint group can be identified as the Foundry Override route spec. When set like this, all constraints that are in that constraint group will be read. If the desired constraint does not exist in or cannot be read from the Foundry Override route spec, then the search goes to the Foundry Rules.

The assignment of a constraint group as the Foundry Override route spec is transient and cannot be saved. For more information, see set_foundry_override, unset_foundry_override, and get_foundry_override.

Foundry Rules

All constraints that are set in the Foundry route spec will be read.

Related Topics

Specialty Routing Constraints

Using Tapers

set_override_rulespec

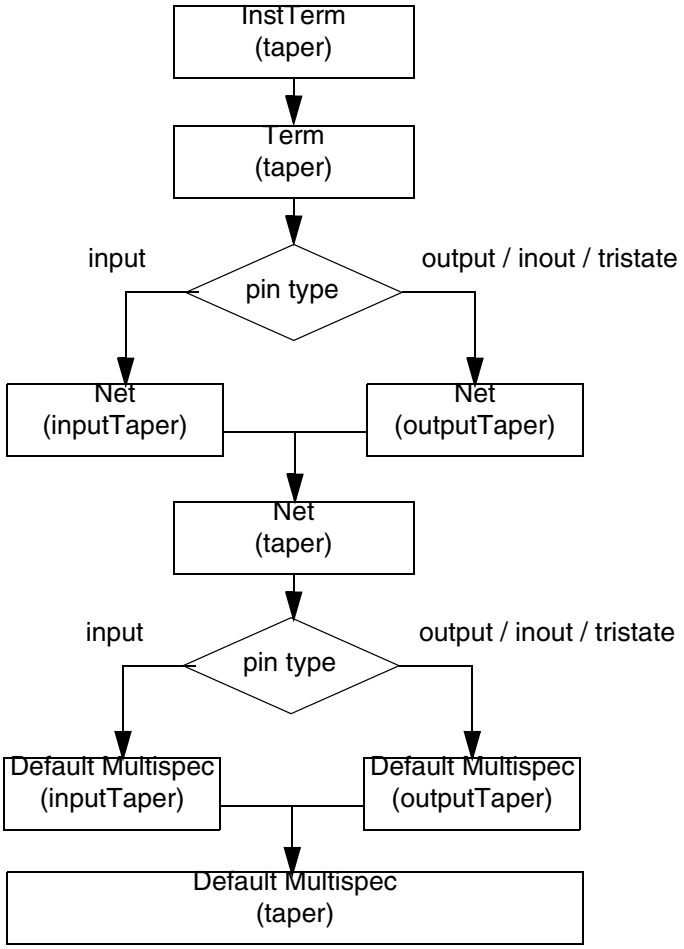
Specialty Routing Constraints

There are special search hierarchies for tapers, shields, and other specialty routing.

Taper Constraints

Constraints that control tapering to or from the terms on a net are determined using the taper hierarchy shown in the following figure. If a `taper` constraint group is not assigned to an instTerm, it will inherit taper constraints from its term's `taper` constraint group, then its net's `inputtaper` or `outputtaper` constraint group, as appropriate, and so on. By default, the `taper` constraint group for the default multispec is the Global net default route spec.

Search Order for Taper Constraints



Constraints searched for:

- `minSpacing`
- `minWidth`
- `minNumCut`
- `validRoutingLayers`
- `validRoutingVias`
- `taperToFirstVia`
- `minTaperWindow`
- `maxTaperWindow`
  (or `oaTaperHalo`)

The default taper constraint group is the global net default.

Shield Constraints

Most of the constraints that control shielding are assigned to the `shield` constraint group for the nets to be shielded, but they can also be assigned to the `shield` group for specific routes

of those nets. This allows you to customize constraints for routes on a net. For example, you could specify a wider width shield wire around a particular route of a net.

Constraint Name	Search Order

`minWidth`
`minSpacing`
`tandemWidth`
`tandemLayerAbove`
`tandemLayerBelow`
	■ `shield` group for the route
	■ `shield` group for the net
	■ `shield` group of the default multispec.

`shareShields`	■ Design route spec
`ignoreShieldingOnLayers`	■ Global net default route spec

Bus, Net Pair, and Matched Length Nets Constraints

The net groups for buses, net pairs and matched length nets are created using create_group. The search order for constraints that control the routing of these specialty net groups is shown below.

Constraint Name	Search Order
`minSpacing`	■ Reflexive constraint for the netGroup determines the spacing between nets within the netGroup ■ TransReflexive constraint for the netGroup determines the spacing between nets in the netGroup and nets outside the netGroup ■ Default constraint group for the net ■ Default constraint group for the netGroup ■ Global net default ■ Design route spec ■ Foundry route spec

Constraint Name	Search Order

`minWidth`
`validRoutingLayers`
`validRoutingVias`
`minNumCut`
	■ Default constraint group for the net
	■ Default constraint group for the netGroup
	■ Global net default
	■ Design route spec
	■ Foundry route spec

`lengthPatternAccordion`
`lengthPatternEndRun`
`lengthPatternOff`
`lengthPatternRWAccordion`
`lengthPatternTrombone`
`routeMaxLength`
`routeMinLength`
	(Matched length net groups only)
	■ Default constraint group for the route
	■ Default constraint group for the net
	■ Default constraint group for the netGroup
	■ Global net default
	■ Design route spec
	■ Foundry route spec

`msMatchPerLayer`
`msTolerance`
(or `matchTolerance`)
	(Matched length net groups only)
	■ Default constraint group for the netGroup
	■ Global net default
	■ Design route spec
	■ Foundry route spec

Related Topics

Controlling Shield Options

Constraint Group Types

create_group

Constraint Group Operators

Constraint groups use an operator to determine how the constraints in the group are applied. There are three constraint group operators:

- Precedence

  Databases based on OpenAccess dataModel 3 or earlier use precedence to determine how multiple constraints in a constraint group apply. Databases based on dataModel 4 or later support a precedence type constraint group operator that duplicates the behavior of the earlier dataModels. The precedence type of constraint group operator requires that the first hard constraint found in the group must be met. If a soft constraint is found, it is enforced, if possible. Otherwise, the next constraint in the group is considered. Only one constraint in the group applies, and the first hard or soft constraint that is met satisfies the constraint group evaluation. If the first hard constraint found cannot be met, no further constraints in the group are considered. Although uncommon, it is possible for constraint groups using the precedence operator to have other groups as members.

- AND

  The AND constraint group operator requires that *all* constraints in the group that *apply to the situation* must be met. Other constraint groups may not be included or nested in an AND constraint group.

- OR

  The OR constraint group operator requires that only *one* constraint in the group that *applies to the situation* must be met. If a hard constraint cannot be met, the next constraint in the group is considered. Other constraint groups are not supported in an OR constraint group.

You must set the constraint group operator when you create the constraint group.

```
create_constraint_group -name s_name -opType {and|or|precedence}
```

If the operator is not specified, precedence is used.

Use the following commands for constraint groups:

- create_constraint_group creates a custom constraint group, route spec, or rule spec.
- assign_constraint_group assigns a constraint group to an object.
- assign_group_group assigns an interchild group-to-group constraint group to two net groups.
- unassign_constraint_group removes the constraint group from an object.

- set_constraint_group assigns route specs to constraint groups for the multispec of a net, a net group, a set of objects, or an area boundary.
- set_default_constraint_group sets a constraint group as the default.

Related Topics

Constraint Objects

Simplified Constraint Search Hierarchy

Applying Constraints to Objects

An object group is a collection of design objects. Use the following commands for object groups:

- **create_group** creates an object group.
- **report_group** outputs information about one or more object groups.

The general procedure for applying constraints to objects (nets, routes, terms, and area boundaries) is as follows:

1. (Optional) Define custom constraints.
2. (Optional) If you want to apply the constraints to a group of objects, create the object group.
3. (Optional) If you are applying the constraints to a new constraint group, create the constraint group.
4. Assign the constraint group. .

Important

You must specify the constraint group type (`implicit`, `default`, `foundry`, `taper`, `inputtaper`, `outputtaper`, `shield`, `transreflexive`, `reflexive`, or `interchild`).

5. For each constraint, do the following:
   a. (Optional) Set the constraint parameters.
   b. Set the constraint. (set_constraint, set_layer_constraint, set_layerpair_constraint, set_layerarray_constraint).

Related Topics

create_group

report_group

define_constraint

create_constraint_group

set_constraint_parameter

Advanced Nodes Constraints

Advanced node constraints and parameters listed in this section are currently for ICADVM use only. For constraints with parameters listed here in the description, the constraint is considered to be advanced node only when used with one or more of the specified parameters.
For IC6.1.x releases, you cannot load a design with any of these constraints or parameters set.

Constraint Category	Constraint	Description
Area	minAreaEdgeLength	Minimum area edge length with `exceptStep` parameter
	oaMinRectArea	Minimum area for rectangular shapes on a layer with `maxWidth` parameter
Clearance	minCenterLineClearance	Minimum clearance from centerline of a layer1 shape to a layer2 shape
	minClusterClearance	Minimum clearance between a group of aligned shapes and a shape on a different layer
	minCornerClearance	Minimum spacing between corners on two different layers with `cornerType`, `otherCornerType`, or `deltaVoltage` parameter

Constraint Category	Constraint	Description
	minCutRoutingClearance	Minimum clearance depends on the cut class of layer1 cut, or the width of a shape which encloses the cut on a given layer with `anyOppositeExtension`, `enclosingLayer`, `enclosingLayerWidth`, `maskOverlap`, `otherExtension`, `parallelEdgeWithin`, `parallelExtension`, or `parallelRunLength` parameters
	minNeighboringShapesClearance	Minimum spacing between two shapes on different layers in presence of the neighboring shapes
Extension	minDualExtension	Minimum extension for a shape on layer1 past a shape on layer2 with `oaSpacingDirection` or `stepSizePair` parameters
	minExtension	Minimum extension for shape on layer1 past a shape on layer2 with `cutClass`, `oaSpacingDirection`, or `exceptEdgeLengthRange` parameters
	minExtensionEdge	Minimum extension on a given edge with `exceptConcaveCornerWithin` or `twoSides` parameters
	minExtensionToCorner	Minimum extension on long side
	minExtensionToCorner	Restricts how close a cut shape can be to the corners of the enclosing metal
	minInnerVertexProximityExtension	Minimum extension for cuts that are near an inner vertex of the enclosing metal
	minNeighborExtension	Minimum extension on two opposite sides of a cut with only one neighbor wire

Constraint Category	Constraint	Description
	minSideExtension	Minimum extension for a short or long side edge of a layer1 shape past a layer2 shape with `exceptEdgeLengthRangeArray` parameter
Length and Width	allowedBoundaryDimensions	Allowed ranges for PR boundary dimensions in the specified direction
	allowedLengthRange	Allowed length ranges for rectangular shapes on a layer
	allowedNeighborOverLayerWidthRange	Allowed neighbor width ranges over (three layer)
	allowedNeighborWidthRange	Allowed neighbor width ranges (single layer)
	allowedWidthRange	Allowed width ranges with `interSpace`, `measurementDirection`, or `withinRange` parameters
	minEndOfLineAdjacentToStep	Minimum length of an end-of-line edge that is adjacent to an edge smaller tjhan a certain length with `concaveCorner` or `adjacentLength` parameter
Miscellaneous	edgeMustCoincide	Conditions for which edges of one layer must coincide with edges of another layer
	rectangularShapeDirection	Requires shapes tro be rectangular with `extendBy`, `widthRangeArray`, `exceptViaLayer`, `exceptViaSize`, or `exceptExactSize` parameter
Overlap	edgeMustOverlap	Edges on layer1 must be overlapped by a shape on layer2.
	minDirectionalOverlap	Required overlap in a given direction between two shapes
	minWireOverlap	Minimum wire overlap

Constraint Category	Constraint	Description
Spacing	endOfLineKeepout	Area to avoid around an end-of-line with `exceptFrom`, `twoSides`, or `colorMask` parameter
	forbiddenEdgePitchRange	Forbidden edge pitch range
	forbiddenProximitySpacing	Forbidden spacing range
	minBoundaryInteriorHalo	Minimum spacing to an enclosing PR boundary with `oaSpacingDirection` or `interSpace` parameter
	minClusterSpacing	Minimum spacing between a group of aligned shapes and other shapes on the same layer
	minEdgeLengthSpacing	Minimum edge length spacing
	minEndOfLineSpacing	Minimum end-of-line spacing with `oaConnectivityType`, `bothWires`, `exactEolWidth`, `extendBy`, `sizeBy`, `oaSpacingDirection`, `widthRangeArray`, `wrongDirSpace`, `diffMask`, or `exceptExactAligned` parameter
	minJointCornerSpacing	Minimum spacing between two facing joints of joint corners with `sameMask` parameter
	minProtrusionSpacing	Minimum protrusion spacing with `excludeSpacing` parameter
	minSideSpacing	Minimum spacing between the shapes on the same layer for a given (short or long) edge
	minSpacing	Minimum spacing with `ignoreShapesInRanges` parameter
	minSpanLengthSpacing	Minimum spacing that is dependent on the parallel run length and span length of two objects

Constraint Category	Constraint	Description
	oaAllowedSpacingRange	Allowed set of spacing ranges between any two shapes on a layer with `distance`, `exceptOtherRanges`, `exceptOverLayer`, `exceptRanges`, `interSpace`, `maxWidth`, `numShapesRange`, `overLayer`, `parallelRunLength`, `widthRangeArray`, or `withinRange` parameters
	shapeRequiredBetweenSpacing	Minimum spacing between shapes on `layer1` that do not require a shape on `layer2` between them
Via Construction	cutClass	Cut class with a `fixedOrientation` parameter
	forbiddenCutClassSpacingRange	Forbidden cut class spacing range
	minAdjacentViaSpacing	Minimum distance between adjacent via cuts with `allCuts`, `cutSizeRangeArray`, or `exceptAllCornerNeighbors` parameters
	minCutClassClearance	Minimum cut class clearance with `nonZeroEnclosure` or `nonCutClassEdgeSpacing` parameter
	minCutClassSpacing	Minimum cut class spacing with `nonCutClassEdgeSpacing` parameter
	minLargeViaArrayCutSpacing	Minimum spacing between via cuts in a large cut array with `maxNumCuts` parameter
	oaMinViaSpacing	Minimum via spacing with `bothCuts`, `enclosingLayer`, `oaSpacingDirection`, `sizeBy`, `viaEdgeTypeParam`, or `exceptExactAligned` parameters

Related Topics

Multi-patterning (MPT) Constraints

Constraint Group Types

Specialty Routing Constraints

Multi-patterning (MPT) Constraints

Multi-patterning constraints listed in the table below specify the minimum required spacing between shapes on the same layer for technologies that require more than one lithography mask for any drawn layer. For constraints in this group, when the `sameMask` parameter is set to `false` (default), the constraint applies to all shapes on the same layer; when the `sameMask` parameter is set to `true`, the constraint applies to shapes on the same mask.

Constraint Description	Constraint	Constraint Parameter
Side-to-side spacing	minSpacing	sameMask true
Corner-to-corner spacing	minSpacing (PRL<0)	sameMask true
End-to-side spacing	minEndOfLineSpacing	sameMask true
End-to-end spacing	minEndOfLineSpacing	sameMask true endToEndSpace otherEndWidth
Via-to-via spacing	oaMinViaSpacing	sameMask true

Pre-coloring constraints listed below can be assigned to critical nets in the schematic view.

Constraint	Constraint Description
msSameMask	Assigns all shapes on the same routing layer to the same mask

Related Topics

Advanced Nodes Constraints

Constraint Group Types

Specialty Routing Constraints

Layer-Purpose Pair Constraints

Many constraints must be applied to layers, but a few can also be applied to *user-defined* layer purpose pairs. The layer purpose pairs must be defined in the `techPurposes` section of the technology file, and only layer purpose pair constraints in the foundry constraint group are checked.

The following constraints can be applied to *user-defined* layer purpose pairs, as well as to layers:

- minSpacing
- minClearance
- minWidth

To apply layer purpose pair constraints, the user-defined purposes must be specified in the `techPurposes` section of the technology file with either `sigType` or `voltageRange` set, and `techVersion ("1.0")` must be specified in the `controls` section of the technology file.

Example

This example sets the following conditions:

- The minimum width for `("Metal1" "HV")` shapes is 0.14, and for all other `Metal1` shapes, it is 0.12.
- The minimum spacing for `("Metal1" "LV")` shapes is 0.4.
- The minimum spacing for `("Metal1" "logic")` shapes is 0.5.
- The minimum spacing for `("Metal1" "HV")` shapes is 1.0.
- The minimum clearance between a `("Metal1" "HV")` shape and a `("Metal1" "LV")` shape is 0.5.
- The minimum clearance between a `("Metal1" "HV")` shape and a `("Metal1" "logic")` shape is 0.7.
- The minimum clearance between a `("Metal1" "HV")` shape and a `V1` shape is 0.8.

The user-defined `logic`, `hv`, and `lv` purposes are specified in the `techPurposes` section of the technology file, and `techVersion ("1.0")` is specified in the `controls` section of the technology file.

```
controls (
   techVersion ("1.0")
)
techPurposes(
;( PurposeName Purpose#   Abbreviation   [Attributes] )
;( ----------- --------   ------------   ------------ )
;User-Defined Purposes:
 ( logic       2000       LGC        'sigType "digital" 'parent "drawing")
 ( HV          2010       HV         'sigType "digital" 'parent "drawing")
 ( LV          2020       LV         'sigType "digital" 'parent "drawing")
) ;techPurposes
```

In the foundry constraint group, the minimum width, spacing, and clearance constraints are specified as follows:

```
set_layer_constraint -constraint minWidth -layer Metal1 -hardness hard \
  -Value 0.12
set_layer_constraint -constraint minWidth -layer Metal1 -purpose HV \
  -hardness hard -Value 0.14
set_layer_constraint -constraint minSpacing -layer Metal1 -purpose LV \
  -hardness hard -Value 0.4
set_layer_constraint -constraint minSpacing -layer Metal1 -purpose logic \
  -hardness hard -Value 0.5
set_layer_constraint -constraint minSpacing -layer Metal1 -purpose HV \
  -hardness hard -Value 1.0
set_layerpair_constraint -constraint minClearance -layer1 Metal1 \
  -purpose1 HV -layer2 Metal1 -purpose2 LV  -hardness hard -Value 0.5
set_layerpair_constraint -constraint minClearance -layer1 Metal1 \
  -purpose1 HV -layer2 Metal1 -purpose2 logic  -hardness hard -Value 0.7
set_layerpair_constraint -constraint minClearance -layer1 Metal1 \
  -purpose1 HV -layer2 V1 -hardness hard -Value 0.8
```

Related Topics

Advanced Nodes Constraints

Multi-patterning (MPT) Constraints

Constraint Group Types

Specialty Routing Constraints

Voltage-Dependent Rule Support

In 45nm and below process nodes, voltage-specific spacing rules, minVoltageSpacing and minVoltageClearance, are defined by the foundry. These constraints can be layer purpose pair-based or net-based.t

To determine the minimum spacing or minimum clearance for an object, the voltage settings are considered in the following precedence:

- If the object is on a layer purpose pair whose voltage has been set, the voltage for the layer purpose pair is used.
- If the object belongs to a net whose voltage has been set, the net's voltage is used.

Layer Purpose Pair VDR Support

Voltage dependent rules can be applied to *user-defined* layer purpose pairs. These layer purpose pairs must be defined in the `techPurposes` section of the technology file, and `techVersion ("1.0")` must be specified in the `controls` section of the technology file.

A voltage swing must be specified for each layer purpose pair, and the voltage-dependent minimum spacing and clearance (minVoltageSpacing and minVoltageClearance, respectively) are set by layer. Only VDR constraints in the foundry constraint group are checked.

This example of Layer Purpose Pair VDR support sets the following conditions:

- `("Metal1" "HV")` shapes have a voltage swing between 0.0 and 3.3.
- `("Metal1" "LV")` shapes have a voltage swing between 0.0 and 1.5.
- The minimum spacing between `Metal1` shapes whose maximum voltage swing is less than 1.5 is 0.06.
- The minimum spacing between `Metal1` shapes whose maximum voltage swing is greater than or equal to 1.5 and less than 3.3 is 0.08.
- The minimum spacing between `Metal1` shapes whose maximum voltage swing is greater than or equal to 3.3 is 0.1.

The user-defined `hv` and `lv` purposes are specified in the `techPurposes` section of the technology file with their associated voltage ranges, and `techVersion ("1.0")` is specified in the `controls` section of the technology file.

```
controls (
   techVersion ("1.0")
)
techPurposes(
;(PurposeName Purpose# Abbreviation)
;(-------------------------------)
;User-defined Purposes:
(hv 13 hv 'sigType "digital" 'parent "drawing" voltageRange (0.0 3.3))
(lv 14 lv 'sigType "digital" 'parent "drawing" voltageRange (0.0 1.5))
)
```

In the foundry constraint group, the voltage-dependent minimum spacing between two shapes on the `Metal1` layer is set using:

```
set_layer_constraint -constraint minVoltageSpacing -layer Metal1 \
  -hardness hard -row_name voltage \
  -FltHeader1DTblValue { 0.0 0.06   1.5 0.08   3.3 0.1 } \
  -row_interpolation snap_down -row_extrapolation {snap_up snap_down}
```

In the table, values are given in pairs with the first value representing the voltage and the second value representing the spacing. To determine the minimum spacing from the table, the maximum voltage swing between shapes is calculated as (`max`(max voltage of each shape) - (`min`(min voltage of each shape))). This voltage is compared with the "`voltage`" values in the table to find the first value that is less than or equal to the calculated voltage swing. The corresponding minimum spacing is required. For this example,

- For a voltage swing >= 0.0 and < 1.5, the minimum spacing is 0.06.
- For a voltage swing >= 1.5 and < 3.3, the minimum spacing is 0.08.
- For a voltage swing >= 3.3, the minimum spacing is 0.1.

Net-based VDR Support

Voltage dependent rules can be applied to nets. The voltage range of the net must be specified using the set_net_voltage Tcl command and the voltage-dependent minimum spacing and clearance (minVoltageSpacing and minVoltageClearance, respectively) are set by layer. Only VDR constraints in the foundry constraint group are checked. For an example, see the description of set_net_voltage.

Related Topics

Layer-Purpose Pair Constraints

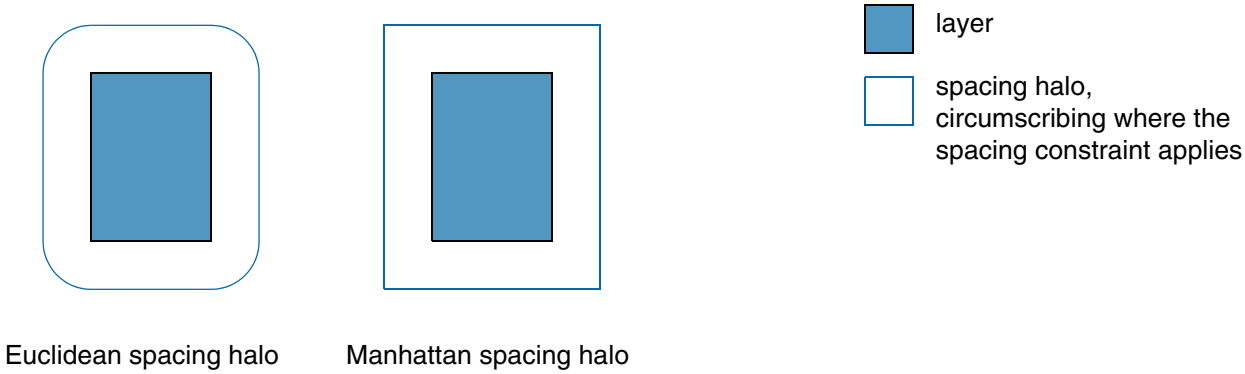
Applying Constraints to Objects

Euclidean and Manhattan Spacing Constraints

By default, spacing constraints apply within the *Euclidean* distance around a shape; the area in which the constraint applies, or the spacing halo, has rounded corners. For some spacing constraints, you can also specify that the distance applied be *Manhattan* instead; the Manhattan spacing halo, which has squared corners, allows a larger spacing area at the corners, as is required for some process rules or constraints. Throughout this manual,

constraints for which you can specify manhattan spacing have an optional `oaDistanceMeasureType` parameter that can be set.

Value	Description
0	Euclidean (rounded corners)
1	Manhattan (squared corners)



Euclidean spacing halo Manhattan spacing halo

Related Topics

Specialty Routing Constraints

Multi-patterning (MPT) Constraints

Advanced Nodes Constraints

Layer-Purpose Pair Constraints

Constraints Summary

The table below lists Virtuoso Space-based Router constraints in alphabetical order and includes the following columns:

- **Constraint** is the name of the constraint with a link to its detailed description.
- **Category** identifies the constraint's function group.
- **Type** identifies the constraint as Simple, Layer, Layer pair, or Layer array.
- **Supported by** indicates whether the constraint is supported by the checker and/or router. Although you can set and get values for the constraints listed here either directly or indirectly, these columns indicate whether those settings are used by during checking and/or routing. Notation is also given for constraints that have been replaced or are no longer supported.

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
135RouteGridOffset	Routing	Layer	no	no
135RouteGridPitch	Routing	Layer	no	no
45RouteGridOffset	Routing	Layer	no	no
45RouteGridPitch	Routing	Layer	no	no
allowedLengthRange	Length and Width	Layer	Advanced Nodes Only	Advanced Nodes Only
allowedNeighborDiffLayerWidthRange	Length and Width	Layer pair	yes	no
allowedNeighborOverLayerWidthRange	Length and Width	Layer array	Advanced Nodes Only	no
allowedNeighborWidthRange	Length and Width	Layer	Advanced Nodes Only	no
allowedWidthRange	Length and Width	Layer	yes	yes
antenna	Antenna	Layer	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
antennaMetalAreaFactor	Antenna	Simple	yes	yes
antennaMetalCutBelowAreaFactor	Antenna	Simple	yes	yes
antennaMetalDiodeAreaFactor	Antenna	Simple	yes	yes
antennaMetalGateAreaFactor	Antenna	Simple	yes	yes
antennaMetalMinusDiodeAreaFactor	Antenna	Simple	yes	yes
antennaViaAreaFactor	Antenna	Simple	yes	yes
antennaViaDiodeAreaFactor	Antenna	Simple	yes	yes
antennaViaGateAreaFactor	Antenna	Simple	yes	yes
boundarySpace	Routing	Layer	no	no
checkSpaceAsMaxXY	Spacing	Layer	obsolete	
CMPAnalysisGrid	CMP	Simple	yes	no
CMPRegionHalo	CMP	Simple	yes	no
cost	Routing	Layer	no	no
crossTalkNeighborIndex	Mixed Signal Routing	Simple	no	yes
cumulativeDAR	Antenna	Simple	yes	yes
cumulativeGAR	Antenna	Simple	yes	yes
cumulativeMetalAntenna	Antenna	Simple	yes	yes
cumulativeSideWall	Antenna	Simple	yes	yes
cumulativeViaAntenna	Antenna	Simple	yes	yes
cumulativeViaDAR	Antenna	Simple	yes	yes
cumulativeViaGAR	Antenna	Simple	yes	yes
cumulativeViaSideWall	Antenna	Simple	yes	yes
cutClass	Via Construction	Layer	yes	yes
cutClassPreference	Routing	Layer	no	yes
defaultHorizontalRouteGridOffset	Routing	Simple	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
defaultHorizontalRouteGridPitch	Routing	Simple	yes	yes
defaultMfgGrid	Routing	Simple	yes	yes
defaultVerticalRouteGridOffset	Routing	Simple	yes	yes
defaultVerticalRouteGridPitch	Routing	Simple	yes	yes
densityFillKeepout	Density	Layer	yes	yes
diagonalShapesAllowed	Miscellaneous	Layer	no	no
diodeAreaRatio	Antenna	Layer	yes	yes
discreteWidth	Length and Width	Layer	obsolete	
edgeMustCoincide	Miscellaneous	Layer pair	Advanced Nodes Only	
effectiveWidth	Length and Width	Layer	yes	yes
endOfLineKeepout	Spacing	Layer	no	yes
errorLayer	Miscellaneous	Layer	yes	no
extendedValidRoutingVias	Routing	Simple	yes	yes
forbiddenCutClassSpacingRange	Via Construction	Layer	Advanced Nodes Only	no
forbiddenEdgePitchRange	Spacing	Layer	Advanced Nodes Only	no
forbiddenProximitySpacing	Spacing	Layer	Advanced Nodes Only	no
gapSpace	Mixed Signal Routing	Layer	obsolete	
gateAreaRatio	Antenna	Layer	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
horizontalPlacementGridOffset	Placement and Alignment	Simple	no	no
horizontalPlacementGridPitch	Placement and Alignment	Simple	no	no
horizontalRouteGridOffset	Routing	Layer	yes	yes
horizontalRouteGridPitch	Routing	Layer	yes	yes
horizontalSymmetryLine	Mixed Signal Routing	Simple	no	yes
ignoreShieldingOnLayers	Mixed Signal Routing	Simple	no	yes
inlineViaPreferred	Routing	Layer pair	yes	yes
inputTaperRule	Mixed Signal Routing	Simple	obsolete	
isDriver	Mixed Signal Routing	Simple	no	yes
isRoutable	Routing	Layer	no	no
keepAlignedShapeAndBoundary	Placement and Alignment	Layer	no	no
keepAlignedShapes	Placement and Alignment	Layer pair	no	no
largeViaClearance	Via Construction	Layer pair	obsolete	
largeViaSpacing	Via Construction	Layer	obsolete	
layerHeight	Routing	Layer	no	no
layerThickness	Routing	Layer	yes	yes
lengthPatternAccordion	Mixed Signal Routing	Simple	no	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
lengthPatternDangle	Mixed Signal Routing	Simple	no	yes
lengthPatternEndRun	Mixed Signal Routing	Simple	no	yes
lengthPatternOff	Mixed Signal Routing	Layer	no	yes
lengthPatternRWAccordion	Mixed Signal Routing	Simple	no	yes
lengthPatternTrombone	Mixed Signal Routing	Simple	no	yes
limitRoutingLayers	Routing	Simple	yes	yes
limitViaThruLayers	Routing	Simple	no	no
matchTolerance	Mixed Signal Routing	Simple	no	yes
maxClearance	Clearance	Layer pair	no	no
maxClusterDistance	Mixed Signal Routing	Layer	no	yes
maxCouplingDiff	Mixed Signal Routing	Simple	no	no
maxCouplingDiffPercent	Mixed Signal Routing	Simple	no	no
maxDensity	Density	Layer	yes	no
maxDensityGradient	CMP	Layer	yes	no
maxDiagonalEdgeLength	Length and Width	Layer	yes	no
maxDiffDensity	Density	Layer	yes	no
maxExtension	Extension	Layer pair	yes	no
maxFloatingArea	Antenna	Simple	yes	no

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
maxGroundCapDiff	Mixed Signal Routing	Simple	no	no
maxGroundCapDiffPercent	Mixed Signal Routing	Simple	no	no
maximumLength	Length and Width	Layer	no	no
maxInterLayerDensityGradient	Density	Simple	yes	no
maxJogLength	Length and Width	Layer	yes	no
maxOxideCopperStepHeight	CMP	Layer	yes	no
maxPickupDistanceAllowed	Routing	Layer	no	yes
maxRoutingDistance	Routing	Layer	no	no
maxRoutingDistanceAllowed	Routing	Layer	no	yes
maxSingleCoupling	Mixed Signal Routing	Simple	no	no
maxSingleCouplingPercent	Mixed Signal Routing	Simple	no	no
maxStressLength	Via Construction	Layer pair	no	no
maxSurfaceHeightRange	CMP	Layer	yes	no
maxTaperWindow	Routing	Simple	yes	yes
maxTapSpacing	Spacing	Layer	no	no
maxThicknessGradient	CMP	Layer	yes	no
maxThicknessGradientPercent	CMP	Layer	yes	no
maxThicknessThreshold	CMP	Layer	yes	no
maxThicknessThresholdPercent	CMP	Layer	yes	no
maxWidth	Length and Width	Layer	yes	yes
mergeSpaceAllowed	Spacing	Layer	no	no

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
mfgGrid	Routing	Layer	yes	yes
minAdjacentViaSpacing	Via Construction	Layer	yes	yes
minArea	Area	Layer	yes	yes
minAreaEdgeLength	Area	Layer	yes	yes
minBoundaryExtension	Extension	Layer	yes	yes
minBoundaryInteriorHalo	Spacing	Layer	yes	yes
minBumpoutEdgeLength	Length and Width	Layer	yes	yes
minCenterLineExtension	Extension	Layer pair	yes	yes
minCenterToCenterSpacing	Spacing	Layer	yes	yes
minClearance	Clearance	Layer pair	yes	yes
minClearanceOverLayer	Clearance	Layer array	no	no
minClusterClearance	Clearance	Layer pair	no	no
minClusterSpacing	Spacing	Layer	Advanced Nodes Only	no
minConcaveCornerExtension	Extension	Layer pair	yes	yes
minConcaveCornerOverlap	Overlap	Layer pair	no	no
minConcaveEdgeLength	Length and Width	Layer	yes	yes
minConvexEdgeLength	Length and Width	Layer	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
minCoupling	Mixed Signal Routing	Simple	no	no
minCouplingPercent	Mixed Signal Routing	Simple	no	no
minCutClassClearance	Via Construction	Layer pair	yes	yes
minCutClassSpacing	Via Construction	Layer	yes	yes
minCutRoutingClearance	Clearance	Layer pair	no	no
minDensity	Density	Layer	yes	yes
minDensityGradient	CMP	Layer	yes	no
minDensityHole	Density	Layer	yes	no
minDiagonalEdgeLength	Length and Width	Layer	yes	yes
minDiagonalSpacing	Spacing	Layer	no	yes
minDiagonalWidth	Length and Width	Layer	yes	yes
minDiffIslandParallelViaSpacing	Spacing	Layer	obsolete	
minDiffPotentialSpacing	Spacing	Layer	no	no
minDirectionalOverlap	Overlap	Layer pair	Advanced Nodes Only	no
minDualEndOfLineExtension	Extension	Layer pair	yes	no
minDualExtension	Extension	Layer pair	yes	yes
minEdgeAdjacentDistance	Length and Width	Layer	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
minEdgeLength	Length and Width	Layer	yes	yes
minEdgeLengthSpacing	Spacing	Layer	Advanced Nodes Only	no
minEdgeMaxCount	Length and Width	Layer	yes	yes
minEnclosedArea	Area	Layer	yes	yes
minEnclosedSpacing	Spacing	Layer	no	no
minEndOfLineCutSpacing	Spacing	Layer pair	no	no
minEndOfLineEdgeExtension	Extension	Layer pair	yes	no
minEndOfLineExtension	Extension	Layer pair	yes	yes
minEndOfLineExtensionSpacing	Spacing	Layer	yes	yes
minEndOfLinePerpSpacing	Spacing	Layer	yes	yes
minEndOfLineSpacing	Spacing	Layer	yes	yes
minEndOfStubSpacing	Spacing	Layer	yes	yes
minExtension	Extension	Layer pair	yes	yes
minExtensionEdge	Extension	Layer pair	yes	yes
minExtensionToCorner	Extension	Layer pair	Advanced Nodes Only	no
minExtensionSpacing	Spacing	Layer	yes	no
minFillPatternSpacing	Density	Layer	yes	no
minFillToFillSpacing	Density	Layer	yes	no

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
minimumLength	Length and Width	Layer	no	yes
minInnerVertexClearance	Clearance	Layer pair	yes	no
minInnerVertexSpacing	Spacing	Layer array	yes	yes
minJogLength	Length and Width	Layer	yes	no
minLargeViaArrayCutSpacing	Via Construction	Layer	yes	yes
minLargeViaArraySpacing	Via Construction	Layer	yes	yes
minLargeViaArrayWidth	Via Construction	Layer	yes	yes
minNeighborExtension	Extension	Layer pair	Advanced Nodes Only	
minNeighboringShapesClearance	Clearance	Layer pair	Advanced Nodes Only	no
minNeighborViaSpacing	Via Construction	Layer	yes	yes
minNotchSpanSpacing	Spacing	Layer	yes	yes
minNumCut	NumCut	Layer	yes	yes
minOneDArrayStructure	Miscellaneous	Layer array	yes	no
minOppositeSpanSpacing	Spacing	Layer	yes	yes
minOuterVertexSpacing	Spacing	Layer pair	yes	yes
minOverlap	Overlap	Layer pair	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
minParallelSpanSpacing	Spacing	Layer	yes	yes
minParallelViaSpacing	Via Construction	Layer	yes	yes
minParallelWithinViaSpacing	Via Construction	Layer	yes	yes
minPerimeter	Length and Width	Layer	yes	yes
minPgFillBoundarySpacing	Density	Layer	yes	no
minPgFillClockSpacing	Density	Layer	yes	no
minPgFillFloatingFillSpacing	Density	Layer	yes	no
minPgFillPgSpacing	Density	Layer	yes	no
minPgFillSignalSpacing	Density	Layer	yes	no
minPgFillSpacing	Density	Layer	yes	no
minProtrudedProximitySpacing	Spacing	Layer	yes	yes
minProtrusionNumCut	NumCut	Layer	yes	yes
minProtrusionSpacing	Spacing	Layer	yes	yes
minProtrusionWidth	Length and Width	Layer	yes	yes
minProximitySpacing	Spacing	Layer	yes	yes
minQuadrupleExtension	Extension	Layer pair	yes	no
minRedundantViaSetback	Via Construction	Layer pair	yes	yes
minSameMetalSharedEdgeViaSpacing	Via Construction	Layer	yes	yes
minSameNetClearance	Clearance	Layer pair	yes	yes
minSameNetSpacing	Spacing	Layer	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
minSideClearance	Clearance	Layer pair	yes	no
minSideExtension	Extension	Layer pair	yes	no
minSideSpacing	Spacing	Layer	no	Advanced Nodes Only
minSize	Length and Width	Layer	yes	yes
minSpacing	Spacing	Layer	yes	yes
minTaperWindow	Routing	Simple	yes	yes
minThicknessGradient	CMP	Layer	yes	no
minThicknessGradientPercent	CMP	Layer	yes	no
minThicknessThreshold	CMP	Layer	yes	no
minThicknessThresholdPercent	CMP	Layer	yes	no
minTouchingDirectionClearance	Clearance	Layer pair	no	no
minTouchingDirectionExtension	Extension	Layer array	no	no
minViaExtension	Via Construction	Layer pair	yes	yes
minViaJointExtension	Extension	Layer pair	yes	no
minVoltageClearance	Clearance	Layer pair	yes	yes
minVoltageSpacing	Spacing	Layer	yes	yes
minWidth	Length and Width	Layer	yes	yes
minWireExtension	Extension	Layer	no	no

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
minWireOverlap	Overlap	Layer pair	Advanced Nodes Only	Advanced Nodes Only
msConnectSupplyDistance	Mixed Signal Routing	Simple	no	yes
msMatchPerLayer	Mixed Signal Routing	Simple	no	yes
msMaxCap	Mixed Signal Routing	Simple	no	yes
msMaxRes	Mixed Signal Routing	Simple	no	yes
msMaxWidth	Mixed Signal Routing	Simple	no	no
msMinCap	Mixed Signal Routing	Simple	no	yes
msMinLengthPatternPitch	Mixed Signal Routing	Layer	no	yes
msMinNumCut	Mixed Signal Routing	Simple	obsolete	
msMinRes	Mixed Signal Routing	Simple	no	yes
msMinSpacing	Mixed Signal Routing	Simple	obsolete	
msMinWidth	Mixed Signal Routing	Simple	obsolete	
msOverhang	Mixed Signal Routing	Simple	obsolete	
msSameMask	Mixed Signal Routing	Simple	yes	yes
msTolerance	Mixed Signal Routing	Simple	no	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
numStrands	Mixed Signal Routing	Simple	no	yes
oaAllowedSpacingRange	Spacing	Layer	yes	yes
oaDefault135RouteGridOffset	Routing	Simple	no	no
oaDefault135RouteGridPitch	Routing	Simple	no	no
oaDefault45RouteGridOffset	Routing	Simple	no	no
oaDefault45RouteGridPitch	Routing	Simple	no	no
oaDummyPolyExtension	Extension	Layer array	no	no
oaGateOrientation	Miscellaneous	Layer	no	no
oaMinEdgeAdjacentLength	Length and Width	Layer	yes	no
oaMinEndOfNotchSpacing	Spacing	Layer	yes	yes
oaMinNotchSpacing	Spacing	Layer	yes	yes
oaMinOrthogonalViaSpacing	Via Construction	Layer	yes	yes
oaMinParallelViaClearance	Via Construction	Layer pair	yes	yes
oaMinRectArea	Area	Layer	yes	yes
oaMinViaClearance	Via Construction	Layer pair	yes	yes
oaMinViaSpacing	Via Construction	Layer	yes	yes
oaPreferredRoutingDirection	Routing	Layer	yes	yes
oaTaperHalo	Routing	Simple	yes	yes
offset	Routing	Layer	yes	yes
orthogonalSnappingLayer	Routing	Layer pair	yes	yes

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
outputTaperRule	Mixed Signal Routing	Simple	obsolete	
pgFillWidth	Length and Width	Layer	yes	
planarTapAllowed	Routing	Simple	no	no
preferredExtensionDirection	Routing	Layer	yes	no
preferredHorizontalDir	Routing	Simple	obsolete	
preferredVerticalDir	Routing	Simple	obsolete	
preferredViaOrigin	Routing	Layer pair	yes	
rectangularLargeViaArraysAllowed	Via Construction	Layer	yes	yes
rectangularShapeDirection	Miscellaneous	Layer	yes	yes
routeMaxLength	Mixed Signal Routing	Simple	no	yes
routeMinLength	Mixed Signal Routing	Simple	no	yes
routeOnGrid	Routing	Simple	no	yes
sameNetLargeViaSpacing	Via Construction	Layer	obsolete	
shapeRequiredClearance	Clearance	Layer pair	yes	no
shapeRequiredBetweenSpacing	Spacing	Layer pair	Advanced Nodes Only	no
shareShields	Mixed Signal Routing	Simple	no	yes
shieldRule	Mixed Signal Routing	Simple	obsolete	

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
shieldWidth	Mixed Signal Routing	Layer	obsolete	
sideWall	Antenna	Layer	yes	
strandSpacing	Mixed Signal Routing	Layer	no	no
strandWidth	Mixed Signal Routing	Layer	no	yes
tandemLayerAbove	Mixed Signal Routing	Layer	no	yes
tandemLayerBelow	Mixed Signal Routing	Layer	no	yes
tandemWidth	Mixed Signal Routing	Layer	no	yes
taperRule	Mixed Signal Routing	Simple	obsolete	
taperToFirstVia	Routing	Simple	yes	yes
TJunctionAllowed	Routing	Simple	no	yes
trimMinSpacing	Spacing	Layer	yes	yes
trimShape	Miscellaneous	Layer	yes	yes
validAntennaDiodes	Antenna	Simple	yes	yes
validFillerCells	Antenna	Simple	yes	yes
validRoutingLayers	Routing	Simple	yes	yes
validRoutingVias	Routing	Simple	yes	yes
validWireEditorVias	Routing	Simple	yes	yes
verticalPlacementGridOffset	Placement and Alignment	Simple	no	yes
verticalPlacementGridPitch	Placement and Alignment	Simple	no	no
verticalRouteGridOffset	Routing	Layer	yes	no

			Supported by	
Constraint	**Category**	**Type**	**Checker**	**Router**
verticalRouteGridPitch	Routing	Layer	yes	yes
verticalSymmetryLine	Mixed Signal Routing	Simple	no	yes
via2ViaPitch	Routing	Layer	no	yes
viaAbove	Routing	Layer	no	yes
viaBarAdjacentSpacing	Via Construction	Layer	obsolete	
viaBelow	Routing	Layer	no	yes
viaEdgeType	Via Construction	Layer	not applicable	not applicable
viaStackingAllowed	Via Construction	Layer pair	yes	yes
viaStackLimit	Via Construction	Simple	yes	yes
viaTapAllowed	Routing	Simple	no	no
wideningMinSplitValue	Wire Widening	Layer	yes	no
wideningTargetWidth	Wire Widening	Layer	yes	no
wire2ViaPitch	Routing	Layer	no	yes
wire2ViaPitchDown	Routing	Layer	no	yes
wire2ViaPitchUp	Routing	Layer	no	yes
wire2WirePitch	Routing	Layer	no	yes
wireExtent	Routing	Layer	no	no
wrongWayOK	Routing	Layer	no	yes

Related Topics

Constraint Definitions

Constraint Group Types

Constraint Definitions

The Virtuoso Space-based Router constraints are grouped by category. Each constraint description includes a definition, a Quick Reference Table, descriptions and examples for parameters and value types. All distance measurements are in user units, and areas in user units^2, unless otherwise noted.

The constraints are grouped into the following categories.

- Antenna Constraints
- Area Constraints
- Clearance Constraints
- CMP Constraints
- Density Constraints
- Extension Constraints
- Length and Width Constraints
- Miscellaneous Constraints
- Mixed-Signal Routing Constraints
- NumCut Constraints
- Overlap Constraints
- Placement and Alignment Constraints
- Routing Constraints
- Spacing Constraints
- Via Construction Constraints
- Wire Widening Constraints

Quick Reference Table

The quick reference table for each constraint lists the following:

- Constraint Type

  Identifies the constraint as one of the following types: Simple, Layer, Layer pair, or Layer array, as described in Constraint Objects.

- Value Types

  Identifies the legal value types for the constraint as described in Constraint Value Types.

- Database Types

  Specifies where the constraint is stored: design or technology database.

- Scope

  Identifies where Virtuoso Space-based Router looks for the value of the constraint, based on the rule spec and object hierarchies described in Scoping Constraints.

Object or Constraint Group	Primary Constraint Group(s)
group2group	InterChild constraint group
route	Default constraint group on the route
Term, InstTerm	Taper constraint group on the Term or InstTerm
net	Default constraint group on the net, also inputTaper, outputTaper, or shield constraint group as appropriate
net group	Default, reflexive, or transreflexive constraint group on the net group
default	Global net default route spec
design	Design route spec of the design database
foundry	Foundry route spec in the Open Access technology database

- Category

  Identifies the constraint category based on the constraint's function.

- Group Operator

  Specifies whether the constraint can be assigned to an AND or OR constraint group.

- OpenAccess API

  For OpenAccess built-in constraints, the OpenAccess constraint definition is given, otherwise `custom`.

Related Topics

Constraints Summary

Constraint Definitions

Constraint Group Types

Tcl Constraint Commands

This topic lists the Tcl constraint commands.

add_constraint_group	append_group	assign_constraint_group
assign_group_group	check_constraint_def	constraint_group_exists
copy_constraint	create_constraint_group	create_group
create_via_variant	define_constraint	destroy_constraint_group
destroy_unused_netoverride_groups	dump_ctu_constraints	dump_oa_constraints
get_constraint	get_constraint_group	get_foundry_override
get_layer_constraint	get_layerarray_constraint	get_layerpair_constraint
get_override_rulespec	remove_constraint_group	report_group
set_constraint	set_constraint_def_check	set_constraint_group
set_constraint_parameter	set_default_constraint_group	set_foundry_override
set_layer_constraint	set_layerarray_constraint	set_layerpair_constraint
set_net_voltage	set_override_rulespec	unassign_constraint_group
unassign_group_group	undefine_constraint	unset_constraint
unset_foundry_override	unset_layer_constraint	unset_layerarray_constraint
unset_layerpair_constraint	unset_override_rulespec	

Related Topics

Manage Constraints

add_constraint_group

```
add_constraint_group
     {-groupName s_groupName | -groupType { design | default | foundry}}
     {-subGroupName s_groupName | -subGroupType {design | default}}
     [-groupDb {tech | design}]
     [-groupTechLib s_techLibName]
     [-subGroupDb {tech | design}]
     [-subGroupTechLib s_techLibNamed]
```

Description

Adds a constraint group as a member of another constraint group.

Arguments

`-groupDB {tech | design}`

Specifies the name of the database that the constraint group belongs to. Defaults to `design`.

`-groupName` *`s_groupName`*

Name of the constraint group to be modified.

`-groupTechLib` *`s_techLibName`*

Name of the technology library to search for the constraint group. This handles inherited technology database structures.

`-groupType {design | default | foundry}`

Specifies the type of the constraint group to be modified.

`default`	Default rule spec
`design`	Design rule spec
`foundry`	Foundry rule spec

`-subGroupDB {tech | design}`

Specifies the name of the database that the member constraint group belongs to. Defaults to `design`.

`-subGroupName` *`s_groupName`*

Names the constraint group to be added as a member.

`-subGroupTechLib` *`s_techLibName`*

Names the technology library to search for the member constraint group. This handles inherited technology database structures.

`-subGroupType {default | design}`

Specifies the type of the constraint group to be added as a member.

`default`	Default rule spec
`design`	Design rule spec

Examples

Adds a member constraint group named `extensions` to the `CG__1` constraint group.

```
add_constraint_group -groupName "CG__1" -subGroupName "extensions"
```

Related Topics

create_constraint_group

append_group

```
append_group
     -name s_groupName
     {-net {s_netName…} | -set d_setObj}
```

Description

Appends one or more nets or nets in a set to the given group. To use this command, you must first create the group.

Arguments

`-name` *`s_groupName`*	Specifies the name of the group to which the nets will be assigned.
`-net` **{***`s_netName`***…}**	Adds nets in the list to the given group.
`-set` *`d_setObj`*	Adds nets in the set to the given group.

Examples

Creates a bus of one net, then adds nets to the net group. The composite net, `bus0`, contains four net members and can be routed as a bus using `bus_route`.

```
create_group -type net_bundle -name bus0 -net {dataIn[0]}
append_group -name bus0 -net {dataIn[1]}
append_group -name bus0 -net {dataIn[2]}
append_group -name bus0 -net {dataIn[3]}
```

Related Topics

bus_route

create_group

assign_constraint_group

```
assign_constraint_group
     -group s_groupName
     {-net s_netName | -set d_setObj | -net_group s_netGroupName
     | -area_boundary s_regionName
     | -term s_termName {[-lib s_libName -cell s_cellName -view s_viewName]
            | [-instance s_instanceName]}}
     [-all_routes]
     [-routes]
     [-verbose]
```

Description

Assigns a constraint group to one of the following:

- A net
- A net group
- The nets, routes, and terms in a set
- A terminal
- A fence or preferred direction region

You can optionally assign a constraint group to the non-taper routes of a net (`-routes`), or to all routes of a net (`-all_routes`).

Arguments

`-all_routes`
: When used with `-net` or `-set`, the constraint group is also applied to all routes of the given net or nets in the set.

`-area_boundary` *`s_regionName`*
: Specifies the name of a region to which the constraint group will be assigned. These regions are created by the `create_soft_fence` and `create_preferred_direction_region` commands.

`-group` *`s_groupName`*
: Specifies the name of constraint group, or rule spec.

`-instance` *`s_instanceName`*
: Used with `-term` to specify the name of the instance that the terminal belongs to.

`-lib` *`s_libName`* `-cell` *`s_cellName`* `-view` *`s_viewName`*

	Used with `-term` to specify the library, cell and view that the terminal belongs to.
`-net` *`s_netName`*	Specifies the name of the net to assign the constraint group to.
`-net_group` *`s_netGroupName`*	
	Specifies the name of the net group to assign the constraint group to.
`-routes`	When used with `-net` or `-set`, the constraint group is also applied to the non-taper routes of the given net or nets in the set.
`-set` *`d_setObj`*	Specifies the set. All nets, routes and terms in the set will be assigned to the constraint group. All other objects in the set are ignored.
`-term` *`s_termName`*	Specifies the name of the terminal to assign the constraint group to. You must give the name of the instance (`-instance`) or the cellview (`-lib`, `-cell`, `-view`) that the terminal belongs to.
`-verbose`	Outputs additional informational messages. By default, these are not included.

Related Topics

create_constraint_group

create_preferred_direction_region

create_soft_fence

assign_group_group

```
assign_group_group
     -net_group1 s_groupName
     -net_group2 s_groupName
     -group_group_spec s_groupGroupName
```

Description

Assigns a group-level constraint group given by the `group_group_spec` argument to the two given net groups.

Arguments

`-group_group_spec` *`s_groupGroupName`*

Specifies the name of the group-level constraint group to assign to the given net groups.

`-net_group1` *`s_groupName`*

Specifies the name of the first net group to assign the group-level constraint group to.

`-net_group2` *`s_groupName`*

Specifies the name of the second net group to assign the group-level constraint group to.

Examples

Creates two net groups, creates a group-level constraint group, and assigns the group-level constraint group to the two net groups.

```
create_group -set $signalNets -name signal_nets -type group
create_group -set $powerNets -name power_nets -type group
create_constraint_group -name gg1 -type group2group
assign_group_group -net_group1 signal_nets -net_group2 power_nets \

 -group_group_spec gg1
```

Related Topics

Tcl Constraint Commands

create_constraint_group

check_constraint_def

```
check_constraint_def
     [-db {tech | design}]
     [-group s_groupName]
     [-constraint s_constraintName]
```

Description

Explicitly checks the constraint definitions in a group or all instances of a constraint, depending on the argument specified in the command.

The check is performed on constraint definition for the validity of values, interdependence of constraints, mandatory constraints (such as `minSpacing`, `minWidth`), correct group, scope of the constraint and other arguments, as defined by the requirements in a constraint.

Some examples of the checks that can be performed are as follows.

- Symmetry in table, such as in the `minOppSpanSpacing` constraint.
- `minProximitySpacing` and `minProtrudedProximitySpacing` are mutually exclusive constraints and cannot be specified together.
- Via constraints should only be specified on `cut` layers.
- `MetalNodeAreaFactor` are only specified on metal layers.

Arguments

`-db {tech | design}`	Specifies the database to search for the constraint group. Default: First design database and then technology database is searched.
`-group`	Specifies the name of the constraint group to be checked.
`-constraint`	Specifies the name of the constraint to be checked.

Examples

```
check_constraint_def constraintgroup1
```

Related Topics

set_constraint_def_check

constraint_group_exists

```
constraint_group_exists
     -name s_groupName
     [-db {tech | design}]
```

Description

Determines whether a constraint group of the given name exists.

Arguments

`-db {tech | design}`	Specifies the database to search for the constraint group. Default: First design database and then technology database is searched.
`-name` *`s_groupName`*	Specifies the name of the constraint group to search for.

Value Returned

`true`	The named constraint group exists.
`false`	The named constraint group does not exist.

Examples

Checks for the existence of the `extensions` constraint group in the design database.

```
constraint_group_exists -name extensions -db design
```

Checks for the existence of the `extensions` constraint group in both the design and technology databases.

```
constraint_group_exists -name extensions
```

Related Topics

add_constraint_group

create_constraint_group

copy_constraint

```
copy_constraint
     -group s_group_name [-to_group s_groupName [-type {design | default | net |
     route | term}]]
     [-constraint s_constraintName [-to_constraint s_constraintName]]
     [-layer {s_layerName …} [-to_layer {s_layerName …}]]
     [-layer1 s_layerName]
     [-layer2 s_layerName]
     [-purpose1 s_purposeName]
     [-purpose2 s_purposeName]
     [-layer_array {s_layerName …}
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
```

Description

Copies constraints and, optionally, creates a new constraint group.

Arguments

`-constraint` *s_constraintName*

Name of the constraint to copy. Defaults to all constraints.

`-group` *s_groupName*

Name of the constraint group or rule spec from which constraints will be copied.

`-layer {`*s_layerName*`…}`

Limits constraints to single layer constraints on the given layer or layers.

`-layer1` *s_layerName*

Limits constraints to layer pair constraints with the given first layer.

`-layer2` *s_layerName*

Limits constraints to layer pair constraints with the given second layer.

`-layer_array {`*s_layerName*`…}`

Limits constraints to layer array constraints with the given list of layers.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Limits constraints to antenna constraints with the given model name.

`-purpose1` *s_purposeName*

Limits constraints to layer or layer pair constraints with the given first purpose.

`-purpose2` *`s_purposeName`*

Limits constraints to layer pair constraints with the given second purpose.

`-to_constraint` *`s_constraintName`*

Name of the constraint to copy to. If given, the constraint must either be a built-in or a predefined custom constraint. If this argument is not given, the original constraint name is used.

`-to_group` *`s_groupName`* Name of the constraint group into which the selected constraints will be copied. If the group does not already exist, it is created. Defaults to the original constraint group.

`-to_layer {`*`s_layerName`* `…}`

Copies constraints to the given layer or layers.

`-type {design | default | net | route | term}`

Specifies the type of constraint group to create.

Examples

Copies all `minSpacing` constraints to `minSameNetSpacing` within the same constraint group.

```
copy_constraint -group CG__1 -constraint minSpacing \
 -to_constraint minSameNetSpacing
```

Copies all `minSpacing` constraints to `minSameNetSpacing` for all constraint groups.

```
foreach routespec [report_rs -all] {copy_constraint -group $routespec \
 -constraint minSpacing -to_constraint minSameNetSpacing}
```

Copies all constraints from `RULE1` to `RULE2`, creating `RULE2`, if necessary.

```
copy_constraint -group RULE1 -to_group RULE2
```

Copies all `Metal1` single layer constraints for `RULE1` to `RULE2`, creating `RULE2`, if necessary.

```
copy_constraint -group RULE1 -to_group RULE2 -layer Metal1
```

Copies all `Metal1` single layer constraints from `RULE1` into equivalent constraints in `RULE2` creating `RULE2`, if necessary.

```
copy_constraint -group RULE1 -to_group RULE2 -layer Metal1 \
 -to_layer {Metal2 Metal3 Metal4 Metal5}
```

Related Topics

Tcl Constraint Commands

create_constraint_group

```
create_constraint_group
     -name s_groupName
     [-type {design | default}]
     [-net s_netName]
     [-techLib s_techLib]
     [-db {tech | design}
     [-opType {and | or | precedence}]
     [-transient [true|false]]
```

Description

Creates a constraint group, or rule spec, and optionally applies the constraints in the constraint group to a net.

Arguments

`-db {tech | design}`	Specifies the name of the database to which the group will belong. Defaults to `design` for all `-type` values except for default, which defaults to `tech`. Also, defaults to tech if `-techLib` is specified.
`-name s_groupName`	Specifies the name for the new constraint group, or rule spec.
`-net s_netName`	Assigns the constraint group as the default constraint group for the named net. Applies all of the constraints in the constraint group to the net. Alternatively, you can omit this argument and issue `set_constraint_group` following this command.
`-opType {and | or | precedence}`	Specifies the name of the constraint group operator for this group. Default: `precedence`
`-techLib s_techLib`	Specifies the technology library to add this constraint group to, in the case that inherited technology libraries are being used. Otherwise, the technology library associated with the design will be used.
`-transient [true|false]`	

When `true`, the new constraint group cannot be saved.

Default: `false`

`-type {design | default}`

Specifies the type of constraint group to create. If this argument is not specified, a user-defined constraint group is created. In most cases, the default rule spec (typically `LEFDefaultRouteSpec`) and the design rule spec (`catenaDesignRules`) will already exist and cannot be overwritten.

`default`	Creates the default rule spec. An error message will be output if this already exists because it cannot be overwritten.
`design`	Creates the design rule spec. An error message will be output if this already exists because it cannot be overwritten.

Examples

Creates a net rule spec, `constGroupA`.

```
create_constraint_group -name constGroupA
```

Related Topics

set_constraint_group

create_group

```
create_group
     {-set d_setObj | -net {s_netName…}}
     [-name s_groupName]
     [-type {net_pair | net_bundle | net_match | net_symmetry | group
     | cross_talk | net_strand | global_bundle}]
```

Description

Creates an object group.

Arguments

`-name` *s_groupName*

Specifies the name for the object group. If this argument is not given, the object group is assigned a name whose prefix identifies the group type, followed by an integer value. For example, for unnamed `net_match` groups, the first group is named `netMatch0`, the second is named `netMatch1`, and so on. The group type/default name prefix pairs are listed below:

`cross_talk`	`crossTalk`
`global_bundle`	`globalBundle`
`group`	`group`
`net_bundle`	`netBundle`
`net_match`	`netMatch`
`net_pair`	`netPair`
`net_strand`	`netStrand`
`net_symmetry`	`netSymmetry`

`-net` **{***s_netName…***}**

Assigns nets in the list to the group.

`-set` *d_setObj*

Assigns the nets in the set to the group.

`-type` *s_groupType*

Specifies the type of group to create. Some Space-based Router and Chip Optimizer features operate on groups of a specific type.

`cross_talk`	Creates a group of nets. Net groups are used by assign_group_group for group-level constraints. For `cross_talk`, neighbor nets are classified as good neighbors, bad neighbors, or neutral for routing purposes by setting the `crosstalkNeighborIndex` constraint.
`global_bundle`	Creates a group of nets that the global router (`global route`) will attempt to route together in common channels.
`group`	Creates a group of nets. Net groups are used by assign_group_group for group-level constraints. This is primarily used to specify a minimum spacing between two groups of nets.
`net_bundle`	Creates a net bundle for bus routing (bus_route).
`net_match`	Creates a group of nets for matching lengths (fix_length).
`net_pair`	Creates a net pair for pair routing (*Preparing for Pair Routing*)
`net_strand`	Identifies nets for strand routing.
`net_symmetry`	Creates a net pair for symmetry routing.

Examples

Creates two net groups which can be used by `assign_group_group` to set group-level constraints, such as minimum spacing, between the two groups of nets.

```
create_group -set $signalNets -name signal_nets -type group
create_group -set $powerNets -name power_nets -type group
```

Creates a group of selected nets. During global route, these nets will be routed together in common channels.

```
create_group -name bundlegroup -type global_bundle -set [get_selection_set]
global_route
```

Related Topics

assign_group_group

create_via_variant

```
create_via_variant
     -name s_viaVariantName
     -via_def_name s_viaDefName
     -cut_width f_userunit
     -cut_height f_userunit
     -cut_spacing {f_x f_y}
     -layer1_enclosure {f_x f_y}
     -layer2_enclosure {f_x f_y}
     [-db {tech | design}]
     [-num_cols i_value]
     [-num_rows i_value]
     [-origin_offset {f_x f_y}
     | -auto_origin_offset
        {positive_layer1 | negative_layer1 | positive_layer2 | negative_layer2}
        -wire_width f_userunit]
     [-layer1_offset {f_x f_y}]
     [-layer2_offset {f_x f_y}]
     [-implant1_enclosure {f_x f_y}]
     [-implant2_enclosure {f_x f_y}]
```

Description

Creates an OpenAccess standard via variant, a permutation of a standard via. The new standard via variant is assigned the given name and represents a pairing of the given via definition in the technology database, and a full set of via parameters.

After creating via variants with this command, you must add the via variant names to the proper constraints in order to use them for routing.

Important

By default, via variants are created in the technology database (`-db tech`) and can be referenced by constraints, such as `validRoutingVias` and `validWireEditorVias`, in the design database or in the technology database. Via variants created in a design database can only be added to constraints in that database and not to constraints in the technology database.

Arguments

`-auto_origin_offset` *s_value*

(Applies only to `1xN` and `Nx1` cut vias with `N`>1) Calculates the origin offset to align the connecting wire with a via metal edge. See Auto Origin Offset Variations for illustrations.

negative_layer1	Via shapes are offset in the negative direction to edge-align the layer 1 via shape with the connecting wire.
negative_layer2	Via shapes are offset in the negative direction to edge-align the layer 2 via shape with the connecting wire.
positive_layer1	Via shapes are offset in the positive direction to edge-align the layer 1 via shape with the connecting wire.
positive_layer2	Via shapes are offset in the positive direction to edge-align the layer 2 via shape with the connecting wire.

`-cut_height` *`f_userunit`*

Height of cut shapes in the via variant.

`-cut_spacing` **{***`f_x f_y`***}**

X and Y spacing around cut shapes for the via variant.

`-cut_width` *`f_userunit`*

Width of cut shapes in the via variant.

`-db {tech | design}` Specifies the name of the database to store the via variant in.

Default is `tech`.

`-implant1_enclosure` **{***`f_x f_y`***}**

X and Y offset of implant1 around the layer1 shape of the via variant.
Default is {0 0}

`-implant2_enclosure` **{***`f_x f_y`***}**

X and Y offset of implant2 around the layer1 shape of the via variant.
Default is {0 0}

`-layer1_enclosure` **{***`f_x f_y`***}**

X and Y enclosure around cut shapes of layer1 for the via variant.

`-layer1_offset {`*`f_x f_y`*`}`

X and Y offset for the lower left corner of the bounding box of the layer1 rectangle defined by the variant. See Layer Offset Variations for example.
Default is {0 0}

`-layer2_enclosure {`*`f_x f_y`*`}`

X and Y enclosure around cut shapes of layer2 for the via variant.

`-layer2_offset {`*`f_x f_y`*`}`

X and Y offset for the lower left corner of the bounding box of the layer2 rectangle defined by the variant. See Layer Offset Variations for examples.
Default is {0 0}

`-name` *`s_viaVariantName`*

Specifies the name for the via variant.

`-num_cols` *`i_value`* Number of cut columns. Default is 1.

`-num_rows` *`i_value`* Number of cut rows. Default is 1.

`-origin_offset {`*`f_x f_y`*`}`

Specifies the X and Y center for the bounding box of the cut matrix defined by the variant. See Origin Offset Variations for examples. Default is {0 0}

`-via_def_name` *`s_viaDefName`*

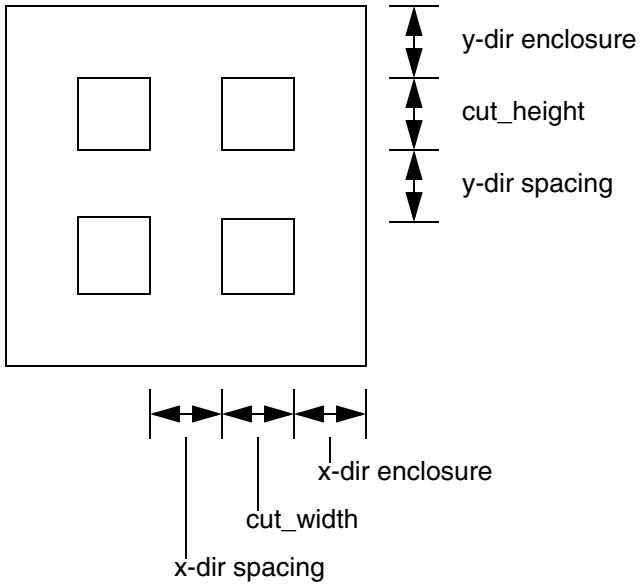
Specifies the name of the standard via definition that the via variant will be a permutation of.

`-wire_width` *`f_userunit`* (Required for `-auto_origin_offset`) Specifies the width of the connecting wire.

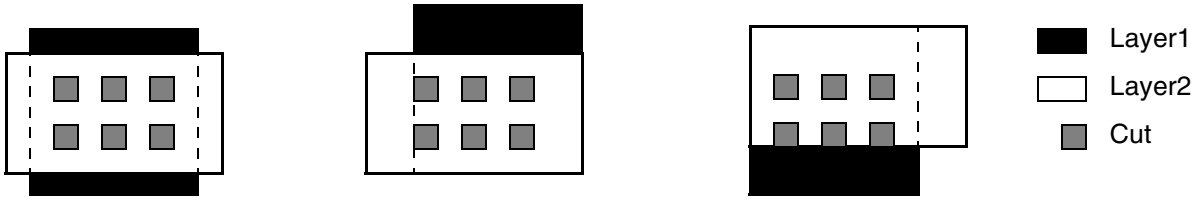
Examples

Creates a standard via variant named `stdvia_m1m2_2x2` in the technology database for the `stdvia_m1m2` via definition in the technology database.

```
create_via_variant -name "stdvia_m1m2_2x2" -via_def_name "stdvia_m1m2" \
 -num_rows 2 -num_cols 2 -cut_width .2 -cut_height .2 \
 -cut_spacing {.1 .2} -layer1_enclosure {.05 .05} -layer2_enclosure {.05 .1}
```



Layer Offset Variations

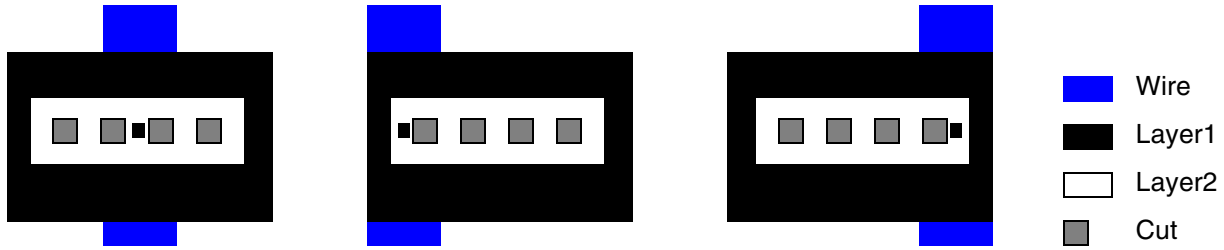


Default: no offset

-layer1_offset {1 1}

-layer2_offset {1 1}

Auto Origin Offset Variations

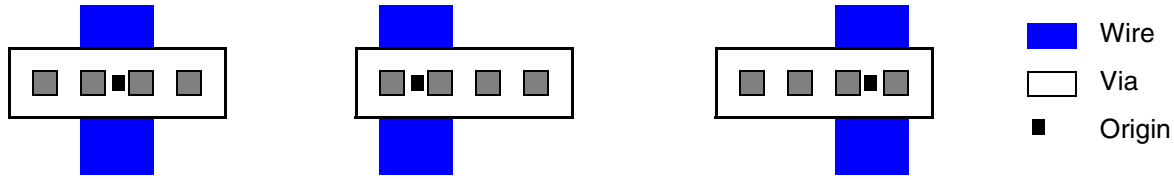


Default: origin in center of all cuts

positive_layer1: edges aligned, shapes offset in the positive direction

negative_layer1: edges aligned, shapes offset in the negative direction

Origin Offset Variations



Default: origin in center of all cuts

-origin_offset {1 0}

-origin_offset {-1 0}

Related Topics

set_constraint

define_constraint

```
define_constraint
     -name s_constraintName
     -type {simple | layer | layerpair | layerarray}
     -allowed_value s_valueType
```

Description

Creates a definition for a constraint. Use this command to create custom constraints. After defining the constraint, set the value using the appropriate `set*constraint` command, then add the constraint to the constraint group for the object that you want the constraint applied to.

Arguments

`-allowed_value` *`s_valueType`*

Indicates the type of value that can be assigned to this constraint. Choices are described in Constraint Value Types.

`-name` *`s_constraintName`*

Specifies the name for the new constraint.

`-type {simple | layer | layerpair}`

Specifies the type of constraint to create.

Related Topics

set_constraint

set_layer_constraint

set_layerarray_constraint

set_layerpair_constraint

destroy_constraint_group

```
destroy_constraint_group
     -name s_groupName
     [-db {tech | design}]
```

Description

Destroys the given constraint group and removes it from any object that references it.

Arguments

`-db {tech | design}`	Specifies the type of database for the constraint group. Default: First design database and then technology database is searched.
`-name s_groupName`	Specifies the name of the constraint group to destroy.

Value Returned

`0`	The named constraint group was successfully destroyed.
`-1`	The named constraint group does not exist or some other error occurred.

Examples

Destroys the `extensions` constraint group of type design if it exists; else destroys from technology database.

```
destroy_constraint_group -name extensions
```

Related Topics

constraint_group_exists

create_constraint_group

destroy_unused_netoverride_groups

```
destroy_unused_netoverride_groups
```

Description

Removes any net override constraint groups that were created in the current session but are not currently referenced by any nets, routes, or other groups. This is useful when `set_preferred_layers` is issued in the session, which creates new constraint groups named `*_netoverride_*`, and then the preferred layers are unset (`unset_preferred_layers`). If the new net override constraint groups are no longer needed, run this command before saving the database to OpenAccess to prevent unnecessary bloating of the database.

Arguments

None

Related *Topics*

set_preferred_layers

unset_preferred_layers

dump_ctu_constraints

```
dump_ctu_constraints
     [-all] | [[-tech][-design][-default]]
     [-members [true|false]]
     [-group s_groupName]
     [-db {tech | design}]
     [-constraint s_constraintName]
     [-name_space {0A2.0 | OA2.2 | default}]
     [-file s_fileName]
     [-report_group [true | false]]
     [{-layer s_layerName
     |{[-layer1 s_layerName] [-layer2 s_layerName]}}
     | -layer_array {s_layerName…}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
```

Description

Outputs Tcl commands for constraints in the current Space-based Router and Chip Optimizer database, including derived and local constraints. The Tcl commands can be cut and pasted into a script or interactively to add or update constraints. By default, all constraints are output to the Transcript area. Options let you output to a file and limit the scope to specific constraints or constraint groups.

Arguments

`-all`	Includes constraints from all constraint groups. This is the default.
`-constraint` *s_constraintName*	Limits output to the specified constraint. By default, all constraints are considered.
`-db {tech|design}`	Specifies the name of the database in which to find the group. Default: First design database and then technology database is searched.
`-default`	Equivalent to the OA 2.0 default route spec. Applies to any routes that do not have a route spec on them or on their net.
`-design`	Includes constraints related to a specific design (`DesignRules`).
`-file` *s_fileName*	Output the Tcl commands to the given file.

`-group` *`s_groupName`*
: Limits output to the specified constraint group. By default all constraint groups are considered.

`-layer` *`s_layerName`*
: Limits constraints to single layer constraints on the given layer.

`-layer1` *`s_layerName`*
: Limits constraints to layer pair constraints with the given first layer.

`-layer2` *`s_layerName`*
: Limits constraints to layer pair constraints with the given second layer.

`-layer_array` **{***`s_layerName`***…}**
: Limits constraints to layer array constraints with the given list of layers.

`-members [true|false]`
: Controls whether member constraints are included. Defaults to `false`.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`
: Limits constraints to antenna constraints with the given model name.

`-name_space {OA2.0 | OA2.2 | default}`
: Specifies the format for the constraint names to output.

  `OA2.0` — Outputs names in OA2.0 format (for example, `minSpaceRule`).

  `OA2.2` — Outputs names in OA2.2 format (for example, `minSpacing`).

  `default` — Outputs names in OA2.0 format for pre-OA2.2 design databases; uses OA2.2 for all others. This is the default.

`-report_group [true | false]`
: Controls whether the `-group` *`s_groupName`* argument is included with each constraint. This is useful for comparing output from different Space-based Router and Chip Optimizer versions. Defaults to `false`.

`-tech`
: Includes constraints from the foundry constraint group.

Examples

Requests Tcl commands for all constraints and shows some of the output.

```
dump_ctu_constraints
Foundry:
  Constraint group CG__1 (id = 1, 116 items):
  #create_constraint_group -name CG__1 -type foundry
    set_layer_constraint -layer PC -constraint minSpacing -hardness hard \
     -value 0.2
    set_layer_constraint -layer PC -constraint minWidth -hardness hard \
     -value 0.2 …
```

Requests Tcl commands for the `Metal1` layer constraints of the `singlestacked` constraint group.

```
dump_ctu_constraints -layer Metal1 -group singlestacked
```

Outputs Tcl commands to a file for all constraints in the foundry and design constraint groups.

```
dump_ctu_constraints -file myconstraints.tcl -tech -design
```

Related Topics

dump_oa_constraints

dump_oa_constraints

```
dump_oa_constraints
     [-file s_fileName]
     [-tech [true | false]]
     [-design [true | false]]
     [-net [true | false]]
     [-route [true | false]]
     [-term [true | false]]
     [-all [true | false]]]
```

Description

Outputs Tcl commands for setting the constraints in the current database to the current values. You can cut and paste these into a script or interactively to add or update constraints. You can send output to the Transcript area (default) or to a file.

Arguments

`-all [true | false]`	Determines whether to include constraints for all supported objects. Defaults to true.
`-design [true | false]`	Determines whether to include constraints for oaDesign objects. Defaults to true.
`-file s_fileName`	Outputs the Tcl commands to the given file.
`-net [true | false]`	Determines whether to include constraints for oaNet objects. Defaults to true.
`-route [true | false]`	Determines whether to include constraints for oaRoute objects. Defaults to true.
`-tech [true | false]`	Determines whether to include constraints for oaTech objects. Defaults to true.
`-term [true | false]`	Determines whether to include constraints for oaTerm objects. Defaults to true.

Examples

Requests constraints for oaTech objects and outputs the information with related Tcl commands to a file.

```
dump_oa_constraints -file myoaTech.tcl -tech
```

Related Topics

dump_ctu_constraints

get_constraint

```
get_constraint
     -constraint s_constraintName
     [-group s_groupName]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-parameter s_parameterName]
     [{-row_value f_value | -row_index i_index}]
     [{-col_value f_value | -col_index i_index}]
```

Description

Returns the value for a simple constraint or a constraint parameter.

Arguments

`-col_index` *i_index*
: Specifies the column index for looking up the constraint value in a 2-D table. Must be in the range of `0` to *number of columns*-1.

`-col_value` *f_value*
: Specifies the column key value for looking up the constraint in a 2-D table. Must be a value in user units, typically a length.

`-constraint` *s_constraintName*
: Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid simple constraint names is output.

`-group` *s_groupName*
: Specifies the name of the constraint group, or rule spec. Defaults to the foundry constraint group for OA constraints, and the design constraint group for all other constraints.

`-hardness {hard | soft | preferred | default}`
: Returns the constraint value for the given setting. By default, the `preferred` value is returned.

  `default`
  : Returns the first value found: hard or soft.

  `hard`
  : Returns the hard value.

  `preferred`
  : Returns the first value found: soft or hard.

  `soft`
  : Returns the soft value.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Returns the constraint value for the given antenna model. This argument is only used with antenna category constraints.

`-parameter` *`s_parameterName`*

Specifies that the value of the named parameter be output, instead of the constraint value. By default, the constraint value is returned.

`-row_index` *`i_index`*

Specifies the row index for looking up the constraint value in a 1-D or 2-D table. Must be in the range of `0` to *`number of rows`*-1.

`-row_value` *`f_value`*

Specifies the row selection value for looking up the constraint value in a 1-D or 2-D table. Must be a value in user units, typically a width.

Value Returned

`value`

Is the value of the specified parameter if `-parameter` is given, otherwise, is the constraint value.

`-1`

The constraint or parameter does not exist or is not set.

Examples

Returns the list of valid simple constraint names, provided a design is already loaded.

```
get_constraint -constraint foo
```

Related Topics

set_constraint

get_constraint_group

```
get_constraint_group
     -type {foundry | default | design}
```

Description

Returns the name of the constraint group for the specified constraint group type.

Arguments

`-type {foundry | design | default}`

Specifies the constraint group type.

Value Returned

s_cgName	Is the name of the constraint group.
-1	The command did not run due to a syntax error or a missing required argument.

Related Topics

Tcl Constraint Commands

get_foundry_override

```
get_foundry_override
```

Description

Returns the name of the constraint group that is the foundry override route spec.

Arguments

None

Value Returned

s_groupName	The name of the foundry override constraint group.
-1	Foundry override rulespec is not set.

Examples

Returns the name of the foundry override constraint group.

```
get_foundry_override
myCG
```

Related Topics

set_foundry_override

unset_foundry_override

get_layer_constraint

```
get_layer_constraint
     -constraint s_constraintName
     -layer s_layerName
     [-purpose s_purposeName]
     [-group s_groupName]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-parameter s_parameterName]
     [{-row_value f_value | -row_index i_index}]
     [{-col_value f_value | -col_index i_index}]
```

Description

Returns the value of the named constraint for the given layer or layer purpose pair.

Arguments

`-col_index` *i_index*
: Specifies the column index to use for looking up the constraint value in a 2-D table. Must be in the range of `0` to *number of columns*-1.

`-col_value` *f_value*
: Specifies the column key value to use for looking up the constraint in a 2-D table. Must be a dimension value, typically a length.

`-constraint` *s_constraintName*
: Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid layer constraint names is output.

`-group` *s_groupName*
: Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard | soft | preferred | default}`
: Returns the constraint value for the given setting. By default, the `preferred` value is returned.

  `default`
  : Returns the first value found: hard or soft.

  `hard`
  : Returns the hard value.

  `preferred`
  : Returns the first value found: soft or hard.

`soft`	Returns the soft value.
`-layer` *s_layerName*	Specifies the name of the layer.
`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`	Returns the constraint value for the given antenna model. This argument is only used with antenna category constraints.
`-parameter` *s_parameterName*	Specifies that the value of the named parameter be output, instead of the constraint value. By default, the constraint value is returned.
`-purpose` *s_purposeName*	With `-layer`, specifies the layer purpose pair.
`-row_index` *i_index*	Specifies the row index for looking up the constraint value in a 1-D or 2-D table. Must be in the range of `0` to *number of rows*-1.
`-row_value` *f_value*	Specifies the row selection value for looking up the constraint value in a 1-D or 2-D table. Must be a dimension value, typically a width.

Value Returned

value	Is the value of the specified parameter if `-parameter` is given, otherwise, is the constraint value.
`-1`	The constraint or parameter does not exist or is not set.

Examples

Returns the list of valid layer constraint names, provided a design is already loaded and layer `Metal1` is a valid layer.

```
get_layer_constraint -layer Metal1 -constraint foo
```

Related Topics

set_layer_constraint

get_layerarray_constraint

```
get_layerarray_constraint
     -constraint s_constraintName
     -layer_array {s_layerName…}
     [-group s_groupName]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-parameter s_parameterName]
     [{-row_value f_value | -row_index i_index}]
     [{-col_value f_value | -col_index i_index}]
```

Description

Returns the value of the named constraint for the given layer array, or list of three or more layers.

Arguments

`-col_index` *i_index*
: Specifies the column index to use for looking up the constraint value in a 2-D table. Must be in the range of `0` to *number of columns*-1.

`-col_value` *f_value*
: Specifies the column key value to use for looking up the constraint in a 2-D table. Must be a dimension value, typically a length.

`-constraint` *s_constraintName*
: Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid layer array constraint names is output.

`-group` *s_groupName*
: Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard | soft | preferred | default}`
: Returns the constraint value for the given setting. By default, the `preferred` value is returned.

  `default`
  : Returns the first value found: hard or soft.

  `hard`
  : Returns the hard value.

  `preferred`
  : Returns the first value found: soft or hard.

`soft`	Returns the soft value.

`-layer_array {`*`s_layerName`*`…}`

Specifies the layers that the constraint applies to. The layer array is a list of three or more layers.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Returns the constraint value for the given antenna model. This argument is only used with antenna category constraints.

`-parameter` *`s_parameterName`*

Specifies that the value of the named parameter be output, instead of the constraint value. By default, the constraint value is returned.

`-row_index` *`i_index`* — Specifies the row index for looking up the constraint value in a 1-D or 2-D table. Must be in the range of `0` to *`number of rows`*-1.

`-row_value` *`f_value`* — Specifies the row selection value for looking up the constraint value in a 1-D or 2-D table. Must be a dimension value, typically a width.

Value Returned

`value`	Is the value of the specified parameter if `-parameter` is given, otherwise, is the constraint value.
`-1`	The constraint or parameter does not exist or is not set.

Examples

Returns the list of valid layer array constraint names, provided a design is already loaded, and layers `Metal1`, `Metal2`, and `Metal3` are valid layers.

```
get_layerarray_constraint -layer_array {Metal1 Metal2 Metal3} -constraint foo
```

Related Topics

set_layerarray_constraint

get_layerpair_constraint

```
get_layerpair_constraint
     -constraint s_constraintName
     -layer1 s_layerName
     -layer2 s_layerName
     [-purpose1 s_purposeName]
     [-purpose2 s_purposeName]
     [-group s_groupName]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-parameter s_parameterName]
     [{-row_value f_value | -row_index i_index}]
     [{-col_value f_value | -col_index i_index}]
     [-symmetric [true | false]]
```

Description

Returns the value of the named constraint for the two given layers or layer purpose pairs.

Arguments

`-col_index` *i_index*
: Specifies the column index to use for looking up the constraint value in a 2-D table. Must be in the range of `0` to *number of columns*-1.

`-col_value` *f_value*
: Specifies the column key value to use for looking up the constraint in a 2-D table. Must be a dimension value, typically a length.

`-constraint` *s_constraintName*
: Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid layer pair constraint names is output.

`-group` *s_groupName*
: Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard | soft | preferred | default}`
: Returns the constraint value for the given setting. By default, the `preferred` value is returned.

  `default`
  : Returns the first value found: hard or soft.

  `hard`
  : Returns the hard value.

`preferred`	Returns the first value found: soft or hard.
`soft`	Returns the soft value.

`-layer1` *`s_layerName`*
Specifies the name of the first layer.

`-layer2` *`s_layerName`*
Specifies the name of the second layer.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`
Returns the constraint value for the given antenna model. This argument is only used with antenna category constraints.

`-parameter` *`s_parameterName`*
Specifies that the value of the named parameter be output, instead of the constraint value. By default, the constraint value is returned.

`-purpose1` *`s_purposeName`*
With `-layer1`, specifies the layer purpose pair for the first layer.

`-purpose2` *`s_purposeName`*
With `-layer2`, specifies the layer purpose pair for the second layer.

`-row_index` *`i_index`*
Specifies the row index for looking up the constraint value in a 1-D or 2-D table. Must be in the range of `0` to *`number of rows`*-1.

`-row_value` *`f_value`*
Specifies the row selection value for looking up the constraint value in a 1-D or 2-D table. Must be a dimension value, typically a width.

`-symmetric [true | false]`
Indicates whether the constraint is symmetric with respect to the layer order. Defaults to true.

Value Returned

value	Is the value of the specified parameter if `-parameter` is given, otherwise, is the constraint value.
`-1`	The constraint or parameter does not exist or is not set.

Examples

Outputs the list of valid layer pair constraint names, provided a design is already loaded and `Metal1` and `Metal2` are valid layers.

```
get_layerpair_constraint -layer1 Metal1 -layer2 Metal2 -constraint foo
```

Related Topics

set_layerpair_constraint

get_override_rulespec

```
get_override_rulespec
```

Description

Returns the name of the override rulespec.

Arguments

None

Value Returned

s_ruleSpec	Is the name of the override rulespec.
-1	Override rulespec is not set.

Examples

Returns the name of the override constraint group.

```
get_override_rulespec
myCG
```

Related Topics

set_override_rulespec

unset_override_rulespec

remove_constraint_group

```
remove_constraint_group
     {-groupName <s_groupName> | -groupType {design | default | foundry}}
     {-subGroupName <s_groupName>} [-groupDb {tech | design}]
     [-groupTechLib <s_techLibName>]
     [-subGroupDb {tech | design}]
     [-subGroupTechLib <s_techLibName>]
```

Description

Removes a subGroup from the first applicable constraint group found.

Arguments

`-groupName` Name of the constraint group to be modified.

`-groupType {design | default | foundry}`

The type of constraint group to be modified. This can be used instead of the required option '`-groupName`'.

`design`	Design rule spec
`default`	Default rule spec
`foundry`	Foundry rule spec

`-subGroupName` Name of the member constraint group that will be removed.

`-groupDb` The name of the database to search for the constraint group. By default, the design database is searched first, followed by the technology database.

`-groupTechLib` The name of the technology library to search for the constraint group. This handles inherited technology database structures.

`-subGroupDb` The name of the database to search for the member constraint group. By default, the design database is searched first, followed by the technology database.

`-subGroupTechLib` The name of the technology library to search for the member constraint group. This handles inherited technology database structures.

Value Returned

`0`	The named member constraint group was successfully removed.
`-1`	The member constraint group does not exist or some other error occurred.

Examples

Removes a member constraint group named `extensions` from the `CG__1` constraint group.

```
remove_constraint_group -groupName "CG__1" -subGroupName "extensions"
```

Related Topics

add_constraint_group

report_group

```
report_group
     {-name s_objectGroupExpr | -set d_setObj | -all}
     [-file s_fileName]
     [-ignore_case [true|false]]
     [-no_wildcard [true|false]]
     [-include_pairs [true|false]]
     [-include_oa_groupdefs [true|false]]
     [-regex_style [true|false]]
```

Description

Outputs information that is about the object groups in a set, given by an expression, or in the entire design, to the transcript area or a file.

Arguments

`-all`
: Outputs information about all the object groups in the design.

`-file` *`s_fileName`*
: Outputs the object group information to the given file.

`-ignore_case [true|false]`
: Performs a case-insensitive search for object group names matching the name given by the `-name` argument. By default, searches are case-sensitive.

`-include_oa_groupdefs [true|false]`
: Outputs information about groups in the OpenAccess database, such as the object types that can be placed in the group, and in which databases the group can be created. By default, this information is not included.

`-include_pairs [true|false]`
: Outputs group-to-group information. For example, two default groups might have a group-to-group interchild relationship. By default, this information is not included.

`-name` *`s_objectGroupExpr`*

Outputs information about the object groups whose names match the expression. If `-regex_style` is set to `true`, the expression is evaluated as a regular expression; otherwise, the expression can include special characters described in *Pattern Matching*.

`-no_wildcard [true|false]`

Disables wildcard processing for the object group name. By default, wildcards can be used.

`-regex_style [true|false]`

Evaluates the `-name` value as a regular expression.

`-set` *`d_setObj`* Outputs information about object groups in the set.

Examples

Issues a `report_group` command and shows output that is returned. In the example, the group found is named `u2245_N4_pair`, a `net_pair` group type with member nets `u2245_N4N` and `u2245_N4`.

```
report_group -name u*
Group: u2245_N4_pair
  Group type: net_pair
  Owner: u2245_N4_pair
  Rule Spec Index: 0
     Net: u2245_N4N
     Net: u2245_N4
```

Related Topics

create_group

set_constraint

```
set_constraint
     -constraint s_constraintName
     [-group s_groupName | -net s_netName]
     [-db {tech | design}]
     [-create [true|false]]
     |-hardness {hard | soft}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-transient [true|false]]
     [-verbose [true | false]]
     { {-BoolValue s_boolean}
     | {-Value f_userunit}
     | {-AreaValue f_userunit}
     | {-CellPathListValue {s_cellview …}}
     | {-DualValue {f_userunit f_userunit}}
     | {-DualValueTbl {{f_userunit f_userunit} ...}}
     | {-IntValue i_value}
     | {-FltValue f_value}
     | {-DblValue f_double}
     | {-LayerValue s_layerName}
     | {[-ViaDefTechLib s_viaTechLib] -ViaDefValue s_viaName}
     | {-StringValue s_value}
     | {-StringAsIntValue s_value}
     | {-LayerArrayValue {s_layerName …}}
     | {-RangeValue s_range}
     | {-RangeArrayValue {s_range …}}
     | {-RangeArray1DTblValue {{ f_width i_count {s_range …}}…}}
     | {-ValueArrayValue {f_userunit …}}
     | {-ViaDefArrayValue {s_viaName …}
     | {{-Int1DTblValue {{f_userunit i_value} …}}
       | {-OneDTblValue {{f_userunit f_userunit} …} [-OneDTblValueB {{f_userunit
     f_userunit} …} ]}
       | {-OneDDualValueTbl {{f_userunit f_userunit f_userunit} ...}}
       | {-OneDDualArrayTblValue {{f_width i_pairCount {{f_ext f_oppExt}…}}…}}
       | {-Dbl1DTblValue {{f_userunit f_value} …}}
       | {-FltHeader1DTblValue {{f_value f_userunit} …}}
       | {-Flt1DTblValue {{f_userunit f_value}…}}
       | {-LngDbl1DTblValue {{f_userunit f_value} …}}
       [-row_name s_name]
       [-row_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
       | linear}}]}
     | {-TblCols {f_userunit …}
       { -Int2DTblValue {{f_userunit {i_value …}} …}
       | -TwoDTblValue {{f_userunit {f_userunit …}} …}
       | -Dbl2DTblValue {{f_userunit {f_value …}} …}}
         [-row_name s_name]
         [-row_interpolation {snap_down | snap_down_inclusive | snap_up
         | snap_up_inclusive | linear}]
```

```
[-row_extrapolation {{snap_down | snap_up | linear}
                     {snap_down | snap_up | linear}}]
[-col_name s_name]
[-col_interpolation {snap_down | snap_down_inclusive | snap_up
| snap_up_inclusive | linear}]
[-col_extrapolation {{snap_down | snap_up | linear}
                     {snap_down | snap_up | linear}}]}
```

Description

Sets a simple constraint value and/or adds a simple constraint to a constraint group.

You must set parameters (set_constraint_parameter) for the constraint prior to issuing this command in order for the parameter(s) to be applied properly.

Arguments

`-AreaValue` *`f_userunit`* Sets the constraint to the value in user units to represent a squared dimension for area values.

`-BoolValue` *`s_boolean`* Sets the constraint to this boolean value.

`-CellPathListValue {`*`s_cellView`* `…}`

Sets the constraint to one or more lib/diodecell/view strings.

`-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`

Indicates how the value is extrapolated for the 2-D table column. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint value for the highest column value if the key is greater than the highest column value.
`snap_up`	Returns the constraint value for the lowest column value if the key is less than the lowest column value.
`linear`	

`-col_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 2-D table column when the column key value does not match a column value but is within the range of the column values.

`snap_down`	Returns the constraint value for the next lower column value.
`snap_down_inclusive`	Returns the constraint value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint value for the next higher row value.
`snap_up_inclusive`	Returns the constraint value for the next higher header value, including when the key matches the lower header value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher values, in proportion to where the key is in that range.

`-col_name` *`s_name`*
: Specifies the name for the 2-D table column. Defaults to `length`.

`-constraint` *`s_constraintName`*
: Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid simple constraint names is output.

`-create [true|false]`
: When `true`, will create a new constraint even if one already exists. This is used for constraints in an AND or OR type constraint group. When `false` (default), if a constraint already exists, its value is overwritten, otherwise the new constraint is created.

`-db {tech|design}`
: Specifies the name of the database in which to find the group, if specified.

  Default: First design database and then technology database is searched.

`-Dbl1DTblValue` **{**`{`*`f_userunit f_value`*`}` **…}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a double precision constraint value.

`-Dbl2DTblValue {{`*f_userunit* `{`*f_value* `…}} …}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one double precision constraint value for each column value given by the `-TblCols` argument.

`-DblValue` *f_value*

Sets the constraint to the double precision value.

`-DualValue {`*f_userunit f_userunit*`}`

Sets the constraint to dual dimensions in user units.

`-DualValueTbl {{`*f_userunit f_userunit*`}…}`

Sets the constraint to the given table of dual dimensions in user units. The table must include an even number of dimensions. For example, `{1.0 1.5 2.0 2.5}` represents two dual dimensions.

`-Flt1DTblValue {{`*f_userunit f_userunit*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes a user unit header value (typically representing width) and a constraint float value.

`-FltHeader1DTblValue {{`*f_value f_userunit*`}…}`

Sets the constraint to the 1-D table with float value headers and values in user units (user units).

`-FltValue` *f_value*

Sets the constraint to the float value.

`-group` *s_groupName*

Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard|soft}`

Indicates the level of hardness or strictness for the constraint:

`hard`
Ground rule or hard setting. Constraint must be met. This is the default and is equivalent to `-hard`.

`soft`	Recommended or better setting. This is equivalent to `-hard false`.

`-Int1DTblValue` **{**{*f_userunit i_value*} **…}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and an integer constraint value.

`-Int2DTblValue` **{**{*f_userunit* {*i_value* **…**}} **…}**

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one integer constraint value for each column value given by the `-TblCols` argument.

`-IntValue` *i_value* — Sets the constraint to the integer value.

`-LayerArrayValue` **{***s_layerName* **…}**

Sets the constraint to the list of layer names.

`-LayerValue` *s_layerName*

Sets the constraint to the layer name.

`-LngDbl1DTblValue` **{**{*f_userunit f_value*} **…}**

Sets the constraint to the 1-D table of paired values. Each pair includes an area value header and a double precision constraint value.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Specifies the antenna model to associate the constraint value with.

`-net` *s_netName* — Adds the constraint to the constraint group of the named net. If the net does not already have a constraint group, one is created.

`-OneDDualArrayTblValue` **{**{*f_width i_pairCount* {{*f_ext f_oppExt*} …}}…**}**

Sets the constraint to the 1-D table of dual dimension arrays. Each set includes a row value, representing width, a count of dual dimension pairs in the array, and the dual dimension array constraint value. The table must be enclosed in braces and cannot include braces. The following example is acceptable,

```
{1 2 1.0 1.2 1.5 1.7 2 1 1.3 1.6}
```

whereas the next example is not acceptable:

```
{1 2 1.0 {1.2 1.5 1.7 2} 1 1.3 1.6}
```

`-OneDDualValueTbl` **{**{*f_userunit f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of dual dimensions. Each pair includes a row value (typically representing width) and a constraint value as a dual dimension in user units. The number of entries enclosed in braces must be a multiple of 3.

`-OneDTblValue` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-OneDTblValueB` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the second 1-D table of paired values for a dual table set. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-RangeArray1DTblValue` **{**{*f_width i_count* {*s_range* ...}}...**}**

Sets the constraint to the table of ranges, indexed by width using the format given in Range Values.

`-RangeArrayValue` **{***s_range* ...**}**

Sets the constraint to a list of ranges using the format given in Range Values.

`-RangeValue` *s_range*

Sets the constraint to a range using the format given in Range Values.

`-row_extrapolation` **{**{snap_down | snap_up | linear} {snap_down | snap_up | linear}**}**

Indicates how the value is extrapolated for the 1-D or 2-D table row. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint value for the highest row value if the key is greater than the highest row value.
`snap_up`	Returns the constraint value for the lowest row value if the key is less than the lowest row value.
`linear`	

`-row_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 1-D or 2-D table row.

`snap_down`	Returns the constraint value for the next lower row value.
`snap_down_inclusive`	Returns the constraint value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint value for the next higher row value.
`snap_up_inclusive`	Returns the constraint value for the next higher header value, including when the key matches the lower header value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher row values, in proportion to where the key is in that range.

`-row_name` *`s_name`* Specifies the name for the 1-D or 2-D table row. Defaults to `width`.

`-StringAsIntValue` *`s_value`*

Sets the constraint to an integer value that is mapped to the given string.

`-StringValue` *`s_string`* Sets the constraint to the string value.

`-TblCols {`*i_length*`…}` Specifies a list of column values for the 2-D table.

`-transient [true|false]`

When `true`, the constraint exists in memory only and will not be saved. When `false` (default), the constraint will be saved.

`-TwoDTblValue {{`*f_userunit* {*f_userunit* …}}…**}**

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one constraint value in user units for each column value given by the `-TblCols` argument.

`-Value` *f_userunit* Sets the constraint to the value in user units.

`-ValueArrayValue {`*f_userunit*`…}`

Sets the constraint to the list of values in user units (user units).

`-verbose [true|false]` Outputs the old and the new settings for the constraint to the Transcript area. By default, the setting messages are not output.

`-ViaDefArrayValue {`*s_viaName* **…}**

Sets the constraint to the list of via definition names.

`-viaDefTechLib` *s_viaTechLib*

Specifies the technology library for the related `ViaDefValue`.

`-ViaDefValue` *s_viaName* Sets the constraint to the given via definition.

Examples

Returns the list of valid simple constraint names, provided a design is already loaded.

```
set_constraint -constraint foo
```

Specifies the list of layers that can be used when routing.

```
set_constraint -constraint validRoutingVias -hard \
 -ViaDefArrayValue {via1 via2 via2ts via3 via3ts via4 via4ts via5 via5ts \
 via6 via3e via4e via5e}
```

Related Topics

get_constraint

set_constraint_parameter

set_constraint_def_check

```
set_constraint_def_check
     -enabled[true | false]
```

Description

Specifies whether all constraint definitions should be validated when set or imported from an OpenAccess database.

Arguments

`-enabled [true | false]`

When set to `true`, all constraint definitions that are set and imported from an OpenAccess database are validated. By default it is set to `true`.

Examples

Causes all constraint definitions to be validated when set and imported from an OpenAccess database.

```
set_constraint_def_check -enabled True
```

Related Topics

check_constraint_def

set_constraint_group

```
set_constraint_group
     [-net s_netName
     | -net_group s_netGroupName
     | -term s_termName {-instance s_instanceName
          | [-lib s_libName -cell s_cellName -view s_viewName]}
     | -set d_setObj
     | -area_boundary s_boundaryName]
     {[-routes]
     [-taper_routes]
     [-default [s_routeSpecName]]
     [-implicit [s_routeSpecName]]
     [-taper [s_routeSpecName]]
     [-inputtaper [s_routeSpecName]]
     [-outputtaper [s_routeSpecName]]
     [-shield [s_routeSpecName]]
     [-reflexive [s_routeSpecName]]
     [-transreflexive [s_routeSpecName]]
     [-interchild [s_routeSpecName]]
     [-nearfarend [s_routeSpecName]]}
```

Description

Assigns constraint groups to the specified constraint group types (Default, Implicit, Taper, InputTaper, OutputTaper, Shielding, Reflexive, TransReflexive, Interchild, NearFarEnd) for the multispec of a net, a net group, a terminal, a set of objects, or an area boundary. By default, the assignment is made to the default multispec. If route spec names are not given with the constraint group type arguments, the current assignments for the respective constraint group types are removed, rather than set.

Arguments

`-area_boundary` *s_boundaryName*

Specifies the name of an area boundary to assign the constraint group(s) to, or remove the assignment from. If neither `-net`, `-net_group`, `-set`, nor `-area_boundary` is given, then the assignments are made to the default multispec.

`-default [`*s_routeSpecName*`]`

Assigns the named constraint group as the Default constraint group for the given net, net group, set of objects, or area boundary. If this argument is given without a route spec name, then the Default constraint group assignment is removed, rather than set.

`-implicit [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the Implicit constraint group. If this argument is given without a route spec name, then the Implicit constraint group assignment is removed, rather than set.

`-inputtaper [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the InputTaper constraint group. If this argument is given without a route spec name, then the OpenAccess InputTaper constraint group assignment is removed, rather than set.

`-instance` *`s_instanceName`*

Specifies the name of the instance to which the terminal (`-term`) belongs.

`-interchild [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the Interchild constraint group. If this argument is given without a route spec name, then the Interchild constraint group assignment is removed, rather than set.

`-lib` *`s_libName`* `-cell` *`s_cellName`* `-view` *`s_viewName`*

Names the library, cell, and view to which the terminal belongs.

`-net` *`s_netName`*

Specifies the name of the net to assign the constraint group(s) to, or remove the assignment from. If neither `-net`, `-net_group`, `-set`, nor `-area_boundary` is given, then the assignments are made to the default multispec.

`-net_group` *`s_netGroupName`*

Specifies the name of the net group to assign the constraint group(s) to, or remove the assignment from. If neither `-net`, `-net_group`, `-set`, nor `-area_boundary` is given, then the assignments are made to the default multispec.

`-outputtaper [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the OutputTaper constraint group. If this argument is given without a route spec name, then the OutputTaper constraint group assignment is removed, rather than set.

`-reflexive [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the Reflexive constraint group. If this argument is given without a route spec name, then the Reflexive constraint group assignment is removed, rather than set.

`-routes`

Assigns the specified non-taper constraint groups to all the non-taper routes on the specified nets.

`-set` *`d_setObj`*

Specifies the set identifier for the set of objects to assign the constraint group(s) to, or remove the assignment from. Applies only to nets, routes, instTerms and area boundaries in the set; all other objects are ignored. If neither `-net`, `-net_group`, `-set`, nor `-area_boundary` is given, then the assignments are made to the default multispec.

`-shield [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the Shielding constraint group. If this argument is given without a route spec name, then the Shielding constraint group assignment is removed, rather than set.

`-taper [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the Taper constraint group. If this argument is given without a route spec name, then the Taper constraint group assignment is removed, rather than set.

`-taper_routes`

Assigns the specified taper constraint groups to all taper routes on the specified nets.

`-term` *`s_termName`*

Name of the terminal to which the constraint group will be assigned. Must be given with either `-instance` or `-lib -cell -view`.

`-transreflexive [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the TransReflexive constraint group. If this argument is given without a route spec name, then the TransReflexive constraint group assignment is removed, rather than set.

`-nearfarend [`*`s_routeSpecName`*`]`

Assigns the named constraint group as the NearFarEnd constraint group for the given nets. The NearFarEnd constraint group specifies constraints that can override the default or implicit rulespec settings for a percentage of the net, based on the nearFarPercentage constraint setting. If this argument is given without a route spec name, then the NearFarEnd constraint group assignment is removed, rather than set.

Examples

Setting a Global Taper Constraint Group

By default, the global net default route spec is used for tapers. The following commands create a new global taper constraint group for pins.

```
# create new constraint group all_taper
create_constraint_group -name all_taper

# assign the new constraint group as the taper constraint group in the default
# multispec
set_constraint_group -taper all_taper

# set constraints for tapering
set_layer_constraint -group all_taper -layer m1 -constraint minWidth \
 -hardness hard -Value 0.17
set_layer_constraint -group all_taper -layer m1 -constraint minSpacing \
 -hardness hard -Value 0.10
set_layer_constraint -group all_taper -layer m2 -constraint minWidth \
 -hardness hard -Value 0.17
set_layer_constraint -group all_taper -layer m2 -constraint minSpacing \
 -hardness hard -Value 0.10
set_layer_constraint -group all_taper -layer m3 -constraint minWidth \
 -hardness hard -Value 0.17
set_layer_constraint -group all_taper -layer m3 -constraint minSpacing \
```

```
 -hardness hard -Value 0.10

# set the oaTaperHalo
set_constraint -group all_taper -constraint oaTaperHalo -Value 5 -hardness hard
```

Setting Input and Output Taper Constraint Groups for Nets

The following commands create unique taper constraint groups for input and output pins of nets, each with different `maxTaperWindow` constraints.

```
# start by creating the output taper constraint group
create_constraint_group -name output_taper

# get the set of nets to apply the taper to
set inoutTaperNets [find_net -name {ionet*}]

# set output_taper as the outputtaper constraint group
set_constraint_group -set $inoutTaperNets -outputtaper output_taper

# set constraints for tapering
set_constraint -group output_taper -constraint maxTaperWindow -Value 10
set_layer_constraint -group output_taper -layer m1 -constraint minWidth \
 -hardness hard -Value 0.17
set_layer_constraint -group output_taper -layer m1 -constraint minSpacing \
 -hardness hard -Value 0.10
set_layer_constraint -group output_taper -layer m2 -constraint minWidth \
 -hardness hard -Value 0.17
set_layer_constraint -group output_taper -layer m2 -constraint minSpacing \
 -hardness hard -Value 0.10
set_layer_constraint -group output_taper -layer m3 -constraint minWidth \
 -hardness hard -Value 0.17
set_layer_constraint -group output_taper -layer m3 -constraint minSpacing \
 -hardness hard -Value 0.10

# next create the input taper constraint group by copying from the output_taper
copy_constraint -group output_taper -to_group input_taper

# set input_taper as the inputtaper constraint group
set_constraint_group -set $inoutTaperNets -inputtaper input_taper

# change the maxTaperWIndow for the inputtaper constraint group
set_constraint -group input_taper -constraint maxTaperWindow -Value 5
```

Setting a Unique Taper Constraint Group for an InstTerm

The following commands create a taper constraint group for a specific instTerm.

```
# next create the term taper constraint group
create_constraint_group -name term_taper

# get the set containing the instTerm
set instTermdout [find_inst_term -name dout -instance_name INST_s22]

# assign the constraint group directly to instTerm.
set_constraint_group -set $instTermdout -taper term_taper
```

```
# set constraints for tapering
set_constraint -group term_taper -constraint maxTaperWindow -Value 5
set_constraint -group term_taper -constraint minTaperWindow -Value 2
set_layer_constraint -group term_taper -layer m1 -constraint minWidth \
 -hardness hard -Value 0.13
set_layer_constraint -group term_taper -layer m1 -constraint minSpacing \
 -hardness hard -Value 0.10
set_layer_constraint -group term_taper -layer m2 -constraint minWidth \
 -hardness hard -Value 0.13
set_layer_constraint -group term_taper -layer m2 -constraint minSpacing \
 -hardness hard -Value 0.10
set_layer_constraint -group term_taper -layer m3 -constraint minWidth \
 -hardness hard -Value 0.13
set_layer_constraint -group term_taper -layer m3 -constraint minSpacing \
 -hardness hard -Value 0.10
```

Related Topics

assign_constraint_group

unassign_constraint_group

set_constraint_parameter

```
set_constraint_parameter
     -name s_constraintName
     { -BoolValue {true | false}
     | -Value f_userunit
     | -IntValue i_value
     | -FltValue f_value
     | -LayerValue s_layerName
     | -StringAsIntValue s_value
     | -StringValue s_value
     | -DualValue {f_userunit f_userunit}
     | -DualValueTbl {{f_userunit f_userunit}…}
     | -OneDDualArrayTblValue {{f_width i_pairCount {{f_ext f_oppExt}…}}…}
     | -ValueArrayValue {f_userunit …}
     | -RangeValue s_range
     | -RangeArrayValue {s_range …}
     | -LayerDualArrayTblValue {s_value i_count {{f_userunit f_userunit}…}}
     | {{-Int1DTblValue {{f_userunit i_value} …}}
       | -OneDTblValue {{f_userunit f_userunit}…}}
       [-row_name s_name]
       [-row_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
       | linear}}]
     | {-TblCols {f_userunit…}
       { -Int2DTblValue {{f_userunit {i_value …}} …}
       | -TwoDTblValue {{f_userunit {f_userunit …}} …}}
       [-row_name s_name]
       [-row_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
       | linear}}]
       [-col_name s_name]
       [-col_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
       | linear}}]}
```

Description

Sets a constraint parameter. This command must be issued to set the parameter before setting the constraint to which it applies.

Arguments

`-BoolValue {true|false}`

Specifies the Boolean value for this constraint parameter.

`-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`

Indicates how the value is extrapolated for the 2-D table column. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint parameter value for the highest column value if the key is greater than the highest column value.
`snap_up`	Returns the constraint parameter value for the lowest column value if the key is less than the lowest column value.
`linear`	

`-col_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 2-D table column when the column key value does not match a column value but is within the range of the column values.

`snap_down`	Returns the constraint parameter value for the next lower column value.
`snap_down_inclusive`	Returns the constraint parameter value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint parameter value for the next higher row value.
`snap_up_inclusive`	Returns the constraint parameter value for the next higher header value, including when the key matches the lower header value.

`linear`	Returns a constraint parameter value that is between the values for the next lower and the next higher values, in proportion to where the key is in that range.

`-col_name` *`s_name`*
Specifies the name for the 2-D table column. Defaults to `length`.

`-DualValue` **{***`f_userunit f_userunit`***}**
Sets the constraint parameter to dual dimensions in user units.

`-DualValueTbl` **{**{*`f_userunit f_userunit`*}**…}**
Sets the constraint parameter to a table of DualValue in user units.

`-FltValue` *`f_value`*
Specifies the float value for this constraint parameter.

`-Int1DTblValue` **{**{*`f_userunit i_value`*} **…}**
Sets the constraint parameter to the 1-D table of paired values. Each pair includes a row value (typically representing width) and an integer constraint value.

`-Int2DTblValue` **{**{*`f_userunit`* {*`i_value`* **…**}} **…}**
Sets a 2-D table of integer values. Each set includes a row value and one integer value for each column value given by the `-TblCols` argument.

`-IntValue` *`i_value`*
Specifies the integer value for this constraint parameter.

`-LayerDualArrayTblValue` **{***`s_layerName i_count`* {{*`f_userunit f_userunit`*}**…**}**}**
Sets a table of *`i_count`* float value pairs for a given setting (*`s_value`*). For example, this is used to set minimum spacing/minimum parallel run length pairs for a given layer.

`-LayerValue` *`s_layerName`*
Specifies the layer name for this constraint parameter.

`-name` *`s_constraintName`*

Specifies the name of the parameter to add to the next constraint. See the tables in Constraint Parameters names of the built-in constraint parameters.

`-OneDDualArrayTblValue {{`*f_width i_pairCount* {{*f_ext f_oppExt*} …}}…**}**

Sets the parameter to the 1-D table of dual dimension arrays. Each set includes a row value, representing width, a count of dual dimension pairs in the array, and the dual dimension array constraint value. The table must be enclosed in braces and cannot include braces. The following example is acceptable,

```
{1 2 1.0 1.2 1.5 1.7 2 1 1.3 1.6}
```

whereas the next example is not acceptable:

```
{1 2 1.0 {1.2 1.5 1.7 2} 1 1.3 1.6}
```

`-OneDTblValue {{`*f_userunit f_userunit*} …**}**

Sets the parameter to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-RangeArrayValue {`*s_range* …**}**

Sets the constraint parameter to a list of ranges using the format given in Range Values.

`-RangeValue` *s_range* Sets the constraint parameter to a range using the format given in Range Values.

`-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`

Indicates how the value is extrapolated for the 1-D table row. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint value for the highest row value if the key is greater than the highest row value.
`snap_up`	Returns the constraint value for the lowest row value if the key is less than the lowest row value.
`linear`	

`-row_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 1-D table row.

`snap_down`	Returns the constraint value for the next lower row value.
`snap_down_inclusive`	Returns the constraint value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint value for the next higher row value.
`snap_up_inclusive`	Returns the constraint value for the next higher header value, including when the key matches the lower header value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher row values, in proportion to where the key is in that range.

`-row_name` *`s_name`*
Specifies the name for the 1-D table row. Defaults to `width`.

`-StringAsIntValue` *`s_value`*
Sets the constraint parameter to an integer value that is mapped to the given string.

`-StringValue` *`s_value`*
Specifies the string value for this constraint parameter.

`-TblCols` **{***`i_width`* **…}**
Specifies a list of column values for the 2-D table.

`-TwoDTblValue` **{**{*`f_userunit`* {*`f_userunit`* …}}…**}**
Sets the constraint parameter to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one constraint value in user units for each column value given by the `-TblCols` argument.

`-Value` *`f_userunit`*
Specifies the value in user units for this constraint parameter.

`-ValueArrayValue` **{***`f_userunit`***…}**

Sets the constraint parameter to the list of values in user units (user units).

Examples

The `minEdgeMaxCount` constraint has required parameter, `maxLength`. The following sets both of these values.

```
set_constraint_parameter -name maxLength -Value 0.3
set_layer_constraint -layer Metal2 -constraint minEdgeMaxCount -IntValue 2
-hardness hard
```

Related Topics

set_constraint

set_layer_constraint

set_layerarray_constraint

set_layerpair_constraint

set_default_constraint_group

```
set_default_constraint_group
     -group s_groupName
```

Description

Sets the default constraint group. This is needed if the database does not set the default or if you want to change the setting.

Arguments

`-group` *`s_groupName`*	Name of the constraint group to use as the default constraint group or routespec. The named constraint group must exist.

Value Returned

`0`	The default constraint group is set to the named group.
`-1`	The default constraint group was not set due to a command syntax error or the named group does not exist.

Examples

Assigns `single` as the default constraint group.

```
set_default_constraint_group -group single
```

Related Topics

Tcl Constraint Commands

set_foundry_override

```
set_foundry_override
     -name s_groupName
```

Description

Sets the foundry override route spec to the named constraint group, or route spec. This effectively inserts the named constraint group into the Simplified Constraint Search Hierarchy between the design and foundry route specs.

A transient design route spec is created and named *<original design route spec name>_<foundry override route spec name>*, containing two *childRuleSpecs* (the original design route spec and the foundry override route spec). For example, if the design route spec is named designRules when the myCG constraint group is identified as the foundry override, then the transient designRules_myCG design route spec is created containing *designRules* and *myCG.*

Arguments

-name *s_groupName*	Sets the named constraint group as the foundry override.

Value Returned

0	The foundry override is set to the named constraint group.
-1	The foundry override was not set. Ensure that the named constraint group exists.

Examples

Sets the foundry override route spec to myCG constraint group.

```
set_foundry_override -name myCG
```

Related Topics

get_foundry_override

unset_foundry_override

set_layer_constraint

```
set_layer_constraint
     -constraint s_constraintName
     -layer s_layerName
     [-purpose s_purposeName]
     [-group s_groupName | -net s_netName]
     [-db {tech | design}]
     [-create [true|false]
     |-hardness {hard | soft}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-transient [true|false]]
     [-verbose [true | false]]
     { {-BoolValue s_boolean}
     | {-Value f_userunit}
     | {-AreaValue f_userunit}
     | {-CellPathListValue {s_cellview …}}
     | {-DualValue {f_userunit f_userunit}}
     | {-DualValueTbl {{f_userunit f_userunit} ...}}
     | {-IntValue i_value}
     | {-FltValue f_value}
     | {-DblValue f_double}
     | {-LayerValue s_layerName}
     | {[-ViaDefTechLib s_viaTechLib] -ViaDefValue s_viaName}
     | {-StringValue s_value}
     | {-StringAsIntValue s_value}
     | {-StringArrayValue "s_value …"}
     | {-LayerArrayValue {s_layerName …}}
     | {-RangeValue s_range}
     | {-RangeArrayValue {s_range …}}
     | {-RangeArray1DTblValue {{ f_width i_count {s_range …}}…}}
     | {-ValueArrayValue {f_userunit …}}
     | {-ViaDefArrayValue {s_viaName …}
     | {{-Int1DTblValue {{f_userunit i_value} …}}
       | {-OneDTblValue {{f_userunit f_userunit} …}
         [-OneDTblValueB {{f_userunit f_userunit} …} ]}
       | {-OneDDualValueTbl {{f_userunit f_userunit f_userunit} ...}}
       | {-OneDDualArrayTblValue {{f_width i_pairCount {{f_ext f_oppExt}…}}…}}
       | {-Dbl1DTblValue {{f_userunit f_value} …}}
       | {-FltHeader1DTblValue {{f_value f_userunit} …}}
       | {-Flt1DTblValue {{f_userunit f_value}…}}
       | {-LngDbl1DTblValue {{f_userunit f_value} …}}
       [-row_name s_name]
       [-row_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up |
      linear}}]}
     | {-TblCols {f_userunit …}
       { -Int2DTblValue {{f_userunit {i_value …}} …}
       | -TwoDTblValue {{f_userunit {f_userunit …}} …}
       | -Dbl2DTblValue {{f_userunit {f_value …}} …}}
```

```
    [-row_name s_name]
    [-row_interpolation {snap_down | snap_down_inclusive | snap_up
  | snap_up_inclusive | linear}]
    [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
|linear}}]
    [-col_name s_name]
    [-col_interpolation {snap_down | snap_down_inclusive | snap_up
  | snap_up_inclusive | linear}]
    [-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
|linear}}]}
```

Description

Sets a single layer or layer purpose pair constraint value and/or adds a single layer or layer purpose pair constraint to a constraint group.

You must set parameters for the constraint prior to issuing this command in order for the parameter(s) to be applied properly.

Arguments

`-AreaValue` *f_userunit*
: Sets the constraint to the value in user units to represent a squared dimension for area values.

`-BoolValue` *s_boolean*
: Sets the constraint to this boolean value.

`-CellPathListValue {`*s_cellView* **…**`}`
: Sets the constraint to one or more lib/diodecell/view strings.

`-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`
: Indicates how the value is extrapolated for the 2-D table column. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

  `snap_down`
  : Returns the constraint value for the highest column value if the key is greater than the highest column value.

  `snap_up`
  : Returns the constraint value for the lowest column value if the key is less than the lowest column value.

  `linear`

`-col_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 2-D table column when the column key value does not match a column value but is within the range of the column values.

`snap_down`	Returns the constraint value for the next lower column value.
`snap_down_inclusive`	Returns the constraint value for the next lower column value, including when the key matches the higher column value.
`snap_up`	Returns the constraint value for the next higher column value.
`snap_up_inclusive`	Returns the constraint value for the next higher column value, including when the key matches the lower column value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher values, in proportion to where the key is in that range.

`-col_name` *`s_name`*
Specifies the name for the 2-D table column. Defaults to `length`.

`-constraint` *`s_constraintName`*
Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid layer constraint names is output.

`-create [true|false]`
When `true`, will create a new constraint even if one already exists. This is used for constraints in an AND or OR type constraint group. When `false` (default), if a constraint already exists, its value is overwritten, otherwise the new constraint is created.

`-db {tech|design}`
Specifies the name of the database to store the constraint in. Default is `design`.

`-Dbl1DTblValue` **{**{*`f_userunit f_value`*} **…}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a double precision constraint value.

`-Dbl2DTblValue` **{**{*f_userunit* {*f_value …*}} **…}**

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one double precision constraint value for each column value given by the `-TblCols` argument.

`-DblValue` *f_value*

Sets the constraint to the double precision value.

`-DualValue` **{***f_userunit f_userunit***}**

Sets the constraint to dual dimensions in user units.

`-DualValueTbl` **{**{*f_userunit f_userunit*}**…}**

Sets the constraint to the given table of dual dimensions in user units. The table must include an even number of dimensions. For example, `{1.0 1.5 2.0 2.5}` represents two dual dimensions.

`-Flt1DTblValue` **{**{*f_userunit f_userunit*}**…}**

Sets the constraint to the 1-D table of paired values. Each pair includes a user unit header value (typically representing width) and a constraint float value.

`-FltHeader1DTblValue` **{**{*f_value f_userunit*}**…}**

Sets the constraint to the 1-D table with float value headers and values in user units (user units).

`-FltValue` *f_value*

Sets the constraint to the float value.

`-group` *s_groupName*

Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard | soft}`

Indicates the level of hardness or strictness for the constraint:

`hard`
Ground rule or hard setting. Constraint must be met. This is the default and is equivalent to `-hard`.

`soft`	Recommended or better setting. This is equivalent to `-hard false`.

`-Int1DTblValue {{`*f_userunit i_value*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and an integer constraint value.

`-Int2DTblValue {{`*f_userunit* `{`*i_value* `…}} …}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one integer constraint value for each column value given by the `-TblCols` argument.

`-IntValue` *i_value* — Sets the constraint to the integer value.

`-layer` *s_layerName* — Specifies the name of the layer.

`-LayerArrayValue {`*s_layerName* `…}`

Sets the constraint to the list of layer names.

`-LayerValue` *s_layerName*

Sets the constraint to the layer name.

`-LngDbl1DTblValue {{`*f_userunit f_value*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes an area value header and a double precision constraint value.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Specifies the antenna model to associate the constraint value with.

`-net` *s_netName* — Adds the constraint to the constraint group of the named net. If the net does not already have a constraint group, one is created.

`-OneDDualArrayTblValue {{`*f_width i_pairCount* {{*f_ext f_oppExt*} …}}…`}`

Sets the constraint to the 1-D table of dual dimension arrays. Each set includes a row value, representing width, a count of dual dimension pairs in the array, and the dual dimension array constraint value. The table must be enclosed in braces and cannot include braces. The following example is acceptable,

```
{1 2 1.0 1.2 1.5 1.7 2 1 1.3 1.6}
```

whereas the next example is not acceptable:

```
{1 2 1.0 {1.2 1.5 1.7 2} 1 1.3 1.6}
```

`-OneDDualValueTbl` **{**{*f_userunit f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of dual dimensions. Each pair includes a row value (typically representing width) and a constraint value as a dual dimension in user units. The number of entries enclosed in braces must be a multiple of 3.

`-OneDTblValue` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-OneDTblValueB` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the second 1-D table of paired values for a dual table set. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-purpose` *s_purposeName*

With `-layer`, specifies the layer purpose pair for the constraint.

`-RangeArray1DTblValue` **{**{*f_width i_count* {*s_range* ...}}...**}**

Sets the constraint to the table of ranges, indexed by width using the format given in Range Values.

`-RangeArrayValue` **{***s_range* ...**}**

Sets the constraint to a list of ranges using the format given in Range Values.

`-RangeValue` *s_range*

Sets the constraint to a range using the format given in Range Values.

`-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`

Indicates how the value is extrapolated for the 1-D or 2-D table row. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint value for the highest row value if the key is greater than the highest row value.
`snap_up`	Returns the constraint value for the lowest row value if the key is less than the lowest row value.
`linear`	

`-row_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 1-D or 2-D table row.

`snap_down`	Returns the constraint value for the next lower row value.
`snap_down_inclusive`	Returns the constraint value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint value for the next higher row value.
`snap_up_inclusive`	Returns the constraint value for the next higher header value, including when the key matches the lower header value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher row values, in proportion to where the key is in that range.

`-row_name` *`s_name`* Specifies the name for the 1-D or 2-D table row. Defaults to `width`.

`-StringAsIntValue` *`s_value`*

Sets the constraint to an integer value that is mapped to the given string.

`-StringValue` *`s_string`* Sets the constraint to the string value.

`-StringArrayValue "`*`s_value`* `…"`

Sets the constraint to the list of string values.

`-TblCols` **{***`i_length`* **…}** Specifies a list of column values for the 2-D table.

`-transient [true|false]`

When `true`, the constraint exists in memory only and will not be saved. When `false` (default), the constraint will be saved.

`-TwoDTblValue` **{**{*`f_userunit`* {*`f_userunit`* …}}…**}**

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one constraint value in user units for each column value given by the `-TblCols` argument.

`-Value` *`f_userunit`* Sets the constraint to the value in user units.

`-ValueArrayValue` **{***`f_userunit`***…}**

Sets the constraint to the list of values in user units (user units).

`-verbose [true|false]` Outputs the old and the new settings for the constraint to the Transcript area. By default, the setting messages are not output.

`-ViaDefArrayValue` **{***`s_viaName`* **…}**

Sets the constraint to the list of via definition names.

`-viaDefTechLib` *`s_viaTechLib`*

Specifies the technology library for the related `ViaDefValue`.

`-ViaDefValue` *`s_viaName`* Sets the constraint to the given via definition.

Examples

Returns the list of valid layer constraint names, provided a design is already loaded.

```
set_layer_constraint -constraint foo
```

Sets the minimum spacing for same net shapes on the `Metal1` layer to 0.18 user units as a hard constraint.

```
set_layer_constraint -layer Metal1 -constraint minSameNetSpacing -hard true -Value
0.18
```

Related Topics

get_layer_constraint

set_constraint_parameter

set_layerarray_constraint

```
set_layerarray_constraint
     -constraint s_constraintName
     -layer_array {s_layerName…}
     [-group s_groupName | -net s_netName]
     [-db {tech | design}]
     [-create [true|false]
     |-hardness {hard | soft}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-transient [true|false]]
     [-verbose [true | false]]
     { {-BoolValue s_boolean}
     | {-Value f_userunit}
     | {-AreaValue f_userunit}
     | {-CellPathListValue {s_cellview …}}
     | {-DualValue {f_userunit f_userunit}}
     | {-DualValueTbl {{f_userunit f_userunit} ...}}
     | {-IntValue i_value}
     | {-FltValue f_value}
     | {-DblValue f_double}
     | {-LayerValue s_layerName}
     | {[-ViaDefTechLib s_viaTechLib] -ViaDefValue s_viaName}
     | {-StringValue s_value}
     | {-StringAsIntValue s_value}
     | {-LayerArrayValue {s_layerName …}}
     | {-RangeValue s_range}
     | {-RangeArrayValue {s_range …}}
     | {-RangeArray1DTblValue {{ f_width i_count {s_range …}}…}}
     | {-ValueArrayValue {f_userunit …}}
     | {-ViaDefArrayValue {s_viaName …}
     | {{-Int1DTblValue {{f_userunit i_value} …}}
       | {-OneDTblValue {{f_userunit f_userunit} …} [-OneDTblValueB {{f_userunit
     f_userunit} …} ]}
       | {-OneDDualValueTbl {{f_userunit f_userunit f_userunit} ...}}
       | {-OneDDualArrayTblValue {{f_width i_pairCount {{f_ext f_oppExt}…}}…}}
       | {-Dbl1DTblValue {{f_userunit f_value} …}}
       | {-FltHeader1DTblValue {{f_value f_userunit} …}}
       | {-Flt1DTblValue {{f_userunit f_value}…}}
       | {-LngDbl1DTblValue {{f_userunit f_value} …}}
       [-row_name s_name]
       [-row_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up |
      linear}}]}
     | {-TblCols {f_userunit …}
       { -Int2DTblValue {{f_userunit {i_value …}} …}
       | -TwoDTblValue {{f_userunit {f_userunit …}} …}
       | -Dbl2DTblValue {{f_userunit {f_value …}} …}}
         [-row_name s_name]
         [-row_interpolation {snap_down | snap_down_inclusive | snap_up
```

```
    | snap_up_inclusive | linear}]
    [-row_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
|linear}}]
    [-col_name s_name]
    [-col_interpolation {snap_down | snap_down_inclusive | snap_up
  | snap_up_inclusive | linear}]
    [-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up
|linear}}]}
```

Description

Sets a layer array constraint value and/or adds a layer array constraint to a constraint group.

You must set parameters for the constraint prior to issuing this command in order for the parameter(s) to be applied properly.

Arguments

`-AreaValue` *f_userunit*
: Sets the constraint to the value in user units to represent a squared dimension for area values.

`-BoolValue` *s_boolean*
: Sets the constraint to this boolean value.

`-CellPathListValue {`*s_cellView* **…**`}`
: Sets the constraint to one or more lib/diodecell/view strings.

`-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`
: Indicates how the value is extrapolated for the 2-D table column. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

  `snap_down`
  : Returns the constraint value for the highest column value if the key is greater than the highest column value.

  `snap_up`
  : Returns the constraint value for the lowest column value if the key is less than the lowest column value.

  `linear`

`-col_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 2-D table column when the column key value does not match a column value but is within the range of the column values.

`snap_down`	Returns the constraint value for the next lower column value.
`snap_down_inclusive`	Returns the constraint value for the next lower column value, including when the key matches the higher column value.
`snap_up`	Returns the constraint value for the next higher column value.
`snap_up_inclusive`	Returns the constraint value for the next higher column value, including when the key matches the lower column value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher values, in proportion to where the key is in that range.

`-col_name` *`s_name`*
: Specifies the name for the 2-D table column. Defaults to `length`.

`-constraint` *`s_constraintName`*
: Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid layer constraint names is output.

`-create [true|false]`
: When `true`, will create a new constraint even if one already exists. This is used for constraints in an AND or OR type constraint group. When `false` (default), if a constraint already exists, its value is overwritten, otherwise the new constraint is created.

`-db {tech|design}`
: Specifies the name of the database to store the constraint in. Default is `design`.

`-Dbl1DTblValue` **{**`{`*`f_userunit f_value`*`}` **…}**
: Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a double precision constraint value.

`-Dbl2DTblValue {{`*f_userunit* `{`*f_value* `…}} …}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one double precision constraint value for each column value given by the `-TblCols` argument.

`-DblValue` *f_value*

Sets the constraint to the double precision value.

`-DualValue {`*f_userunit f_userunit*`}`

Sets the constraint to dual dimensions in user units.

`-DualValueTbl {{`*f_userunit f_userunit*`}…}`

Sets the constraint to the given table of dual dimensions in user units. The table must include an even number of dimensions. For example, `{1.0 1.5 2.0 2.5}` represents two dual dimensions.

`-Flt1DTblValue {{`*f_userunit f_userunit*`}…}`

Sets the constraint to the 1-D table of paired values. Each pair includes a user unit header value (typically representing width) and a constraint float value.

`-FltHeader1DTblValue {{`*f_value f_userunit*`}…}`

Sets the constraint to the 1-D table with float value headers and values in user units (user units).

`-FltValue` *f_value*

Sets the constraint to the float value.

`-group` *s_groupName*

Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard|soft}`

Indicates the level of hardness or strictness for the constraint:

`hard` — Ground rule or hard setting. Constraint must be met. This is the default and is equivalent to `-hard`.

`soft` — Recommended or better setting. This is equivalent to `-hard false`.

`-Int1DTblValue {{`*f_userunit i_value*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and an integer constraint value.

`-Int2DTblValue {{`*f_userunit* `{`*i_value* `…}} …}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one integer constraint value for each column value given by the `-TblCols` argument.

`-IntValue` *i_value* Sets the constraint to the integer value.

`-layer_array {`*s_layerName* `…}`

Specifies the layers for the constraint.

`-LayerArrayValue {`*s_layerName* `…}`

Sets the constraint to the list of layer names.

`-LayerValue` *s_layerName*

Sets the constraint to the layer name.

`-LngDbl1DTblValue {{`*f_userunit f_value*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes an area value header and a double precision constraint value.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Specifies the antenna model to associate the constraint value with.

`-net` *s_netName* Adds the constraint to the constraint group of the named net. If the net does not already have a constraint group, one is created.

`-OneDDualArrayTblValue {{`*f_width i_pairCount* `{{`*f_ext f_oppExt*`}` `…}}…}`

Sets the constraint to the 1-D table of dual dimension arrays. Each set includes a row value, representing width, a count of dual dimension pairs in the array, and the dual dimension array constraint value. The table must be enclosed in braces and cannot include braces. The following example is acceptable,

```
{1 2 1.0 1.2 1.5 1.7 2 1 1.3 1.6}
```

whereas the next example is not acceptable:

```
{1 2 1.0 {1.2 1.5 1.7 2} 1 1.3 1.6}
```

`-OneDDualValueTbl` **{**{*f_userunit f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of dual dimensions. Each pair includes a row value (typically representing width) and a constraint value as a dual dimension in user units. The number of entries enclosed in braces must be a multiple of 3.

`-OneDTblValue` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-OneDTblValueB` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the second 1-D table of paired values for a dual table set. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-RangeArray1DTblValue` **{**{*f_width i_count* {*s_range* ...}}...**}**

Sets the constraint to the table of ranges, indexed by width using the format given in Range Values.

`-RangeArrayValue` **{***s_range* ...**}**

Sets the constraint to a list of ranges using the format given in Range Values.

`-RangeValue` *s_range*

Sets the constraint to a range using the format given in Range Values.

`-row_extrapolation` **{**{snap_down | snap_up | linear} {snap_down | snap_up | linear}**}**

Indicates how the value is extrapolated for the 1-D or 2-D table row. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint value for the highest row value if the key is greater than the highest row value.
`snap_up`	Returns the constraint value for the lowest row value if the key is less than the lowest row value.
`linear`	

`-row_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 1-D or 2-D table row.

`snap_down`	Returns the constraint value for the next lower row value.
`snap_down_inclusive`	Returns the constraint value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint value for the next higher row value.
`snap_up_inclusive`	Returns the constraint value for the next higher header value, including when the key matches the lower header value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher row values, in proportion to where the key is in that range.

`-row_name` *`s_name`* Specifies the name for the 1-D or 2-D table row. Defaults to `width`.

`-StringAsIntValue` *`s_value`*

Sets the constraint to an integer value that is mapped to the given string.

`-StringValue` *`s_string`* Sets the constraint to the string value.

`-TblCols {`*`i_length`* `…}` Specifies a list of column values for the 2-D table.

`-transient [true|false]`

When `true`, the constraint exists in memory only and will not be saved. When `false` (default), the constraint will be saved.

`-TwoDTblValue {{`*`f_userunit`* {*`f_userunit`* …}}…`}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one constraint value in user units for each column value given by the `-TblCols` argument.

`-Value` *`f_userunit`* Sets the constraint to the value in user units.

`-ValueArrayValue {`*`f_userunit`*`…}`

Sets the constraint to the list of values in user units (user units).

`-verbose [true|false]` Outputs the old and the new settings for the constraint to the Transcript area. By default, the setting messages are not output.

`-ViaDefArrayValue {`*`s_viaName`* `…}`

Sets the constraint to the list of via definition names.

`-viaDefTechLib` *`s_viaTechLib`*

Specifies the technology library for the related `ViaDefValue`.

`-ViaDefValue` *`s_viaName`* Sets the constraint to the given via definition.

Examples

Returns the list of valid layer array constraint names, provided a design is already loaded.

```
set_layerarray_constraint -constraint foo
```

Related Topics

get_layerarray_constraint

set_constraint_parameter

set_layerpair_constraint

```
set_layerpair_constraint
     -constraint s_constraintName
     -layer1 s_layerName
     -layer2 s_layerName
     [-purpose1 s_purposeName]
     [-purpose2 s_purposeName]
     [-symmetric]
     [-group s_groupName | -net s_netName]
     [-db {tech | design}]
     [-create [true|false]
     [-hardness {hard | soft}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-transient [true|false]]
     [-verbose [true | false]]
     { {-BoolValue s_boolean}
     | {-Value f_userunit}
     | {-AreaValue f_userunit}
     | {-CellPathListValue {s_cellview …}}
     | {-DualValue {f_userunit f_userunit}}
     | {-DualValueTbl {{f_userunit f_userunit} ...}}
     | {-IntValue i_value}
     | {-FltValue f_value}
     | {-DblValue f_double}
     | {-LayerValue s_layerName}
     | {[-ViaDefTechLib s_viaTechLib] -ViaDefValue s_viaName}
     | {-StringValue s_value}
     | {-StringAsIntValue s_value}
     | {-LayerArrayValue {s_layerName …}}
     | {-RangeValue s_range}
     | {-RangeArrayValue {s_range …}}
     | {-RangeArray1DTblValue {{ f_width i_count {s_range …}}…}}
     | {-ValueArrayValue {f_userunit …}}
     | {-ViaDefArrayValue {s_viaName …}
     | {{-Int1DTblValue {{f_userunit i_value} …}}
       | {-OneDTblValue {{f_userunit f_userunit} …}
         [-OneDTblValueB {{f_userunit f_userunit} …} ]}
       | {-OneDDualValueTbl {{f_userunit f_userunit f_userunit} ...}}
       | {-OneDDualArrayTblValue {{f_width i_pairCount {{f_ext f_oppExt}…}}…}}
       | {-Dbl1DTblValue {{f_userunit f_value} …}}
       | {-FltHeader1DTblValue {{f_value f_userunit} …}}
       | {-Flt1DTblValue {{f_userunit f_value}…}}
       | {-LngDbl1DTblValue {{f_userunit f_value} …}}
       [-row_name s_name]
       [-row_interpolation {snap_down | snap_down_inclusive | snap_up
       | snap_up_inclusive | linear}]
       [-row_extrapolation {{snap_down | snap_up | linear}
        {snap_down | snap_up |  linear}}]}
     | {-TblCols {f_userunit …}
       { -Int2DTblValue {{f_userunit {i_value …}} …}
```

```
| -TwoDTblValue {{f_userunit {f_userunit …}} …}
| -Dbl2DTblValue {{f_userunit {f_value …}} …}}
  [-row_name s_name]
  [-row_interpolation {snap_down | snap_down_inclusive | snap_up
  | snap_up_inclusive | linear}]
  [-row_extrapolation {{snap_down | snap_up | linear}
  {snap_down | snap_up |linear}}]
  [-col_name s_name]
  [-col_interpolation {snap_down | snap_down_inclusive | snap_up
| snap_up_inclusive | linear}]
  [-col_extrapolation {{snap_down | snap_up | linear}
  {snap_down | snap_up |linear}}]}
```

Description

Sets a double layer/layer purpose pair constraint value and/or adds a double layer/layer purpose pair constraint to a constraint group.

You must set parameters (set_constraint_parameter) for the constraint prior to issuing this command in order for the parameter(s) to be applied properly.

Arguments

`-AreaValue f_userunit` Sets the constraint to the value in user units to represent a squared dimension for area values.

`-BoolValue s_boolean` Sets the constraint to this boolean value.

`-CellPathListValue {s_cellView …}`

Sets the constraint to one or more lib/diodecell/view strings.

`-col_extrapolation {{snap_down | snap_up | linear} {snap_down | snap_up | linear}}`

Indicates how the value is extrapolated for the 2-D table column. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down` Returns the constraint value for the highest column value if the key is greater than the highest column value.

`snap_up` Returns the constraint value for the lowest column value if the key is less than the lowest column value.

`linear`

`-col_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 2-D table column when the column key value does not match a column value but is within the range of the column values.

`snap_down`	Returns the constraint value for the next lower column value.
`snap_down_inclusive`	Returns the constraint value for the next lower column value, including when the key matches the higher column value.
`snap_up`	Returns the constraint value for the next higher column value.
`snap_up_inclusive`	Returns the constraint value for the next higher column value, including when the key matches the lower column value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher values, in proportion to where the key is in that range.

`-col_name` *`s_name`* Specifies the name for the 2-D table column. Defaults to `length`.

`-constraint` *`s_constraintName`*

Specifies the name of the constraint. If you include this argument without a constraint name, the list of valid layer pair constraint names is output.

`-create [true|false]` When `true`, will create a new constraint even if one already exists. This is used for constraints in an AND or OR type constraint group. When `false` (default), if a constraint already exists, its value is overwritten, otherwise the new constraint is created.

`-db {tech|design}` Specifies the name of the database to store the constraint in. Default is `design`.

`-Dbl1DTblValue` **{**{*`f_userunit f_value`*} **…}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a double precision constraint value.

`-Dbl2DTblValue {{`*f_userunit* `{`*f_value* `…}} …}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one double precision constraint value for each column value given by the `-TblCols` argument.

`-DblValue` *f_value*

Sets the constraint to the double precision value.

`-DualValue {`*f_userunit f_userunit*`}`

Sets the constraint to dual dimensions in user units.

`-DualValueTbl {{`*f_userunit f_userunit*`}…}`

Sets the constraint to the given table of dual dimensions in user units. The table must include an even number of dimensions. For example, `{1.0 1.5 2.0 2.5}` represents two dual dimensions.

`-Flt1DTblValue {{`*f_userunit f_userunit*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes a user unit header value (typically representing width) and a constraint float value.

`-FltHeader1DTblValue {{`*f_value f_userunit*`}…}`

Sets the constraint to the 1-D table with float value headers and values in user units (user units).

`-FltValue` *f_value*

Sets the constraint to the float value.

`-group` *s_groupName*

Specifies the name of the constraint group, or rule spec. Defaults to the foundry rule spec for OA constraints, and the design rule spec for all other constraints.

`-hardness {hard | soft}`

Indicates the level of hardness or strictness for the constraint:

`hard`
: Ground rule or hard setting. Constraint must be met. This is the default and is equivalent to `-hard`.

`soft`	Recommended or better setting. This is equivalent to `-hard false`.

`-Int1DTblValue {{`*f_userunit i_value*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and an integer constraint value.

`-Int2DTblValue {{`*f_userunit* `{`*i_value* `…}} …}`

Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one integer constraint value for each column value given by the `-TblCols` argument.

`-IntValue` *i_value* — Sets the constraint to the integer value.

`-layer1` *s_layerName* — Specifies the name of the first layer.

`-layer2` *s_layerName* — Specifies the name of the second layer.

`-LayerArrayValue {`*s_layerName* `…}`

Sets the constraint to the list of layer names.

`-LayerValue` *s_layerName*

Sets the constraint to the layer name.

`-LngDbl1DTblValue {{`*f_userunit f_value*`} …}`

Sets the constraint to the 1-D table of paired values. Each pair includes an area value header and a double precision constraint value.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Specifies the antenna model to associate the constraint value with.

`-net` *s_netName* — Adds the constraint to the constraint group of the named net. If the net does not already have a constraint group, one is created.

`-OneDDualArrayTblValue {{`*f_width i_pairCount* `{{`*f_ext f_oppExt*`} …}}…}`

Sets the constraint to the 1-D table of dual dimension arrays. Each set includes a row value, representing width, a count of dual dimension pairs in the array, and the dual dimension array constraint value. The table must be enclosed in braces and cannot include braces. The following example is acceptable,

```
{1 2 1.0 1.2 1.5 1.7 2 1 1.3 1.6}
```

whereas the next example is not acceptable:

```
{1 2 1.0 {1.2 1.5 1.7 2} 1 1.3 1.6}
```

`-OneDDualValueTbl` **{**{*f_userunit f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of dual dimensions. Each pair includes a row value (typically representing width) and a constraint value as a dual dimension in user units. The number of entries enclosed in braces must be a multiple of 3.

`-OneDTblValue` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the 1-D table of paired values. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-OneDTblValueB` **{**{*f_userunit f_userunit*} ...**}**

Sets the constraint to the second 1-D table of paired values for a dual table set. Each pair includes a row value (typically representing width) and a constraint value in user units.

`-purpose1` *s_purposeName*

With `-layer1`, specifies the first layer purpose pair for the constraint.

`-purpose2` *s_purposeName*

With `-layer2`, specifies the second layer purpose pair for the constraint.

`-RangeArray1DTblValue` **{**{*f_width i_count* {*s_range* ...}}...**}**

Sets the constraint to the table of ranges, indexed by width using the format given in Range Values.

`-RangeArrayValue` **{***s_range* ...**}**

Sets the constraint to a list of ranges using the format given in Range Values.

`-RangeValue` *`s_range`* Sets the constraint to a range using the format given in Range Values.

`-row_extrapolation` **{**`{snap_down | snap_up | linear} {snap_down | snap_up | linear}`**}**

Indicates how the value is extrapolated for the 1-D or 2-D table row. The first entry applies to keys below the range of the table, and second entry applies to keys above the range of the table.

`snap_down`	Returns the constraint value for the highest row value if the key is greater than the highest row value.
`snap_up`	Returns the constraint value for the lowest row value if the key is less than the lowest row value.
`linear`	

`-row_interpolation {snap_down | snap_down_inclusive | snap_up | snap_up_inclusive | linear}`

Indicates how the value is interpolated for the 1-D or 2-D table row.

`snap_down`	Returns the constraint value for the next lower row value.
`snap_down_inclusive`	Returns the constraint value for the next lower header value, including when the key matches the higher header value.
`snap_up`	Returns the constraint value for the next higher row value.
`snap_up_inclusive`	Returns the constraint value for the next higher header value, including when the key matches the lower header value.
`linear`	Returns a constraint value that is between the values for the next lower and the next higher row values, in proportion to where the key is in that range.

`-row_name` *`s_name`*
: Specifies the name for the 1-D or 2-D table row. Defaults to `width`.

`-StringAsIntValue` *`s_value`*
: Sets the constraint to an integer value that is mapped to the given string.

`-StringValue` *`s_string`*
: Sets the constraint to the string value.

`-symmetric`
: Indicates that `layer1` and `layer2` are interchangeable and implies that the same constraint value that is set for the given layer values is equivalent to the constraint value when the layer values are reversed. For example, the constraint value for "`-layer1 Metal1 -layer V1`" is equivalent to the constraint value for "`-layer1 V1 -layer Metal1`". By default, layer pair constraints are not symmetric.

`-TblCols` **{***`i_length`* **…}**
: Specifies a list of column values for the 2-D table.

`-transient [true|false]`
: When `true`, the constraint exists in memory only and will not be saved. When `false` (default), the constraint will be saved.

`-TwoDTblValue` **{**{*`f_userunit`* {*`f_userunit`* …}}…**}**
: Sets the constraint to the 2-D table of ordered sets. Each set includes a row value (typically representing width) and one constraint value in user units for each column value given by the `-TblCols` argument.

`-Value` *`f_userunit`*
: Sets the constraint to the value in user units.

`-ValueArrayValue` **{***`f_userunit`***…}**
: Sets the constraint to the list of values in user units (user units).

`-verbose [true|false]`
: Outputs the old and the new settings for the constraint to the Transcript area. By default, the setting messages are not output.

`-ViaDefArrayValue` **{***`s_viaName`* **…}**
: Sets the constraint to the list of via definition names.

`-viaDefTechLib` *`s_viaTechLib`*

	Specifies the technology library for the related `ViaDefValue`.
`-ViaDefValue` *`s_viaName`*	Sets the constraint to the given via definition.

Examples

Returns the list of valid layer pair constraint names, provided a design is already loaded.

```
set_layerpair_constraint -constraint foo
```

Related Topics

get_layerpair_constraint

set_constraint_parameter

set_net_voltage

```
set_net_voltage
     -net s_netName | -set d_setObj | -default
     -min_voltage f_voltage
     -max_voltage f_voltage
```

Description

Specifies the voltage swing for an individual net or set of nets, or as the default for all nets whose voltage values are not explicitly set. Use this command with minVoltageSpacing and minVoltageClearance constraints to specify the minimum spacing and clearance, based on the net's voltage swing.

Arguments

`-default`	Specifies that the given values should be used as the default for all nets whose voltage values are not explicitly set.
`-max_voltage` *`f_voltage`*	Specifies the maximum voltage for the net(s).
`-min_voltage` *`f_voltage`*	Specifies the minimum voltage for the net(s).
`-net` *`s_netName`*	Specifies that the given voltage swing be used for the given net.
`-set` *`d_setObj`*	Specifies that the given voltage swing be used for nets in the given set.

Value Returned

0	The command was successful.
-1	The command failed due to a syntax error.

Examples

This example sets the following:

- The voltage range for net `VDD6p0` is 0.0 (min) to 6.0 (max).
- The voltage range for net `VDD1p2` is 0.0 (min) to 1.2 (max).
- The voltage range for all nets that are not explicitly set is 0.0 (min) to 2.5 (max).

The following Tcl commands set the voltage swings for the nets in this example:

```
set_net_voltage -max_voltage 2.5 -min_voltage 0.0 -default \
set_net_voltage -max_voltage 6.0 -min_voltage 0.0 -set [find_net \
 -name \"VDD6p0\"]
set_net_voltage -max_voltage 1.2 -min_voltage 0.0 -set [find_net \
 -name \"VDD1p2\"]
```

For `Metal1` and `Metal2` in this example,

- For voltage swings >= 0.0 and < 2.5, the minimum spacing is 0.1.
- For voltage swings >= 2.5 and < 6.0, the minimum spacing is 0.2.
- For voltage swings >= 6.0, the minimum spacing is 0.3.

The voltage-dependent minimum spacing for `Metal1` and `Metal2` is specified by the following:

```
set_layer_constraint -constraint minVoltageSpacing -layer Metal1 \
 -hardness hard -row_name voltage \
 -FltHeader1DTblValue { 0.0 0.1  2.5 0.2  6.0 0.3 } \
 -row_interpolation snap_down -row_extrapolation {snap_up snap_down}
set_layer_constraint -constraint minVoltageSpacing -layer Metal2 \
 -hardness hard -row_name voltage \
 -FltHeader1DTblValue { 0.0 0.1  2.5 0.2  6.0 0.3 } \
 -row_interpolation snap_down -row_extrapolation {snap_up snap_down}
```

For `Metal1` and `V1` shapes in this example,

- For voltage swings >= 0.0 and < 2.5, the minimum clearance is 0.1.
- For voltage swings >= 2.5 and < 6.0, the minimum clearance is 0.22.
- For voltage swings >= 6.0, the minimum clearance is 0.35.

For `Metal2` and `V1` shapes in this example,

- For voltage swings >= 0.0 and < 2.5, the minimum clearance is 0.1.
- For voltage swings >= 2.5 and < 6.0, the minimum clearance is 0.23.
- For voltage swings >= 6.0, the minimum clearance is 0.36.

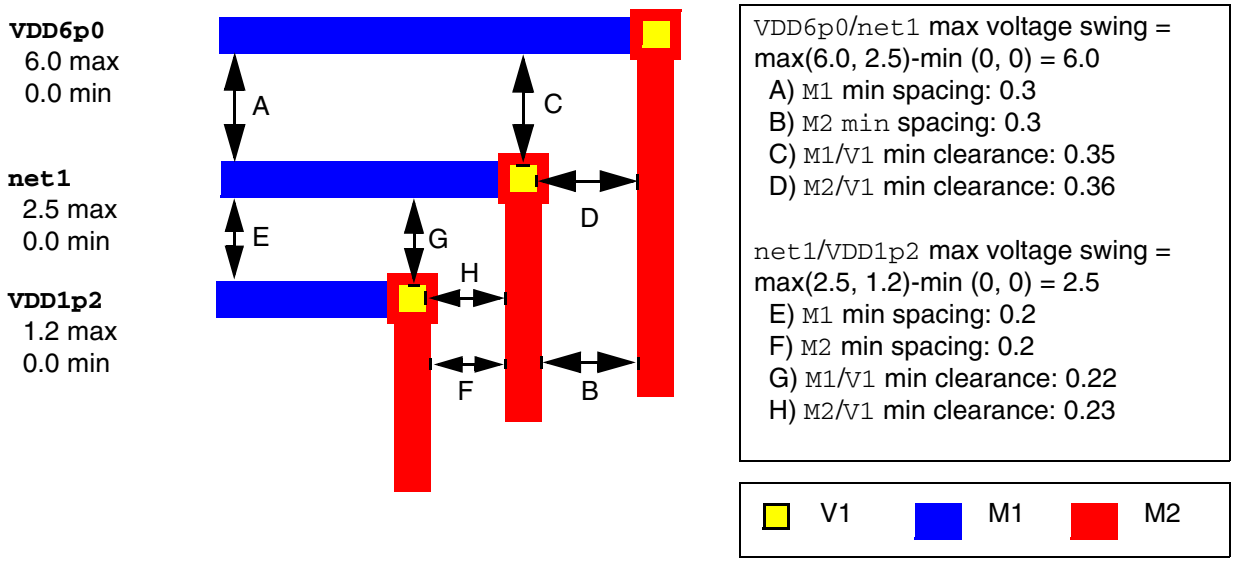
The voltage-dependent minimum clearance between `Metal1` and `V1` shapes and between Metal2 and V1 shapes is specified by the following:

```
set_layerpair_constraint -constraint minVoltageClearance -layer1 Metal1 \
 -layer2 V1 -symmetric true -hardness hard -row_name voltage \
 -FltHeader1DTblValue { 0.0 0.1  2.5 0.22  6.0 0.35 } \
 -row_interpolation snap_down -row_extrapolation {snap_up snap_down}
set_layerpair_constraint -constraint minVoltageClearance -layer1 Metal2 \
 -layer2 V1 -symmetric true -hardness hard -row_name voltage \
```

```
-FltHeader1DTblValue { 0.0 0.1  2.5 0.23  6.0 0.36 } \
-row_interpolation snap_down -row_extrapolation {snap_up snap_down}
```

The following figure shows a graphical example of nets `VDD6p0`, `net1`, and `VDD1p2`. Since voltages were not explicitly set for `net1` in the Tcl script, it inherits the default minimum and maximum voltages, 0.0 and 2.5, respectively.

Net-based VDR Example



Related Topics

Tcl Constraint Commands

minVoltageSpacing

minVoltageClearance

set_override_rulespec

```
set_override_rulespec
     -name s_ruleSpec
```

Description

Sets the specified rulespec as the override rulespec and enables the override rule lookups. The override rulespec is a temporary setting that places the specified override at the top of the list for all rule lookups. This value is not stored persistently.

Arguments

`-name` *`s_ruleSpec`*	Specifies the name of the rulespec to set as the override rulespec.

Examples

Sets the `single` route spec as the override rulespec.

```
set_override_rulespec -name single
```

Related Topics

get_override_rulespec

unset_override_rulespec

unassign_constraint_group

```
unassign_constraint_group
     -net s_netName | -set d_setObj | -net_group s_netGroupName
     | -area_boundary s_regionName
     | -term s_termName {[-lib s_libName -cell s_cellName -view s_viewName]
                        | [-instance s_instanceName]}
     [-verbose]
```

Description

Removes a constraint group assignment from a net, a net group, a fence or preferred direction region, an instance, or from nets, routes and terms in a set. The constraint group is not deleted, even if it is no longer assigned to any objects.

Arguments

`-area_boundary` *s_regionName*

Removes the constraint group assignment from the named fence or preferred direction region.

`-instance` *s_instanceName*

Used with `-term` to specify the name of the instance that the terminal belongs to.

`-lib` *s_libName* `-cell` *s_cellName* `-view` *s_viewName*

Used with `-term` to specify the library, cell and view that the terminal belongs to.

`-net` *s_netName*

Specifies the name of the net to remove the constraint group from.

`-net_group` *s_netGroupName*

Specifies the name of the net group to remove the constraint group from.

`-set` *d_setObj*

Specifies the set. Removes any existing constraint group assignment from the nets, routes and terms in the set. All other objects in the set are ignored.

`-term` *s_termName*

Specifies the name of the terminal to remove the constraint group from. You must give the name of the instance (`-instance`) or the cellview (`-lib`, `-cell`, `-view`) that the terminal belongs to.

`-verbose`	Outputs additional informational messages. By default, these are not included.

Related Topics

assign_constraint_group

unassign_group_group

```
unassign_group_group
     -net_group1 s_netGroupName
     -net_group2 s_netGroupName
```

Description

Removes a group-level constraint group assignment previously set by `assign_group_group`.

Arguments

`-net_group1 s_groupName`

Specifies the name of the first net group.

`-net_group2 s_groupName`

Specifies the name of the second net group.

Related Topics

assign_group_group

undefine_constraint

```
undefine_constraint
     -name s_constraintName
```

Description

Removes a constraint from all constraint groups and removes its definition.

Arguments

`-name` *`s_constraintName`*

Specifies the name of the constraint to remove and undefine.

Related Topics

define_constraint

unset_constraint

```
unset_constraint
     -constraint s_constraintName
     {-group s_groupName | -all}
     [-allMembers [true|false]]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-verbose [true|false]]
```

Description

Selectively removes a simple constraint or value from a specific constraint group or from all constraint groups.

Arguments

`-all` Removes the constraint from all constraint groups.

`-allMembers [true|false]`

When set `true`, removes all members of the constraint from the given group (`-group`) or from all groups (`-all`). This is particularly useful for removing all members from an AND or OR group.

Default: `(false)` Removes only the first member of the constraint.

`-constraint` *`s_constraintName`*

Specifies the name of the constraint to remove.

`-group` *`s_groupName`* Specifies the group from which to remove the constraint.

`-hardness {hard | soft | preferred | default}`

Removes the constraint value for the given enforcement setting. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).

`default`	Removes the first value found: hard or soft.
`hard`	Removes the hard value.
`preferred`	Removes the first value found: soft or hard.
`soft`	Removes the soft value.

`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`	Removes the constraint value for the given antenna model. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).
`-verbose [true|false]`	When set to `true`, the constraint name(s) and group(s) are output.

Related Topics

set_constraint

unset_foundry_override

`unset_foundry_override`

Description

Unsets the foundry override route spec. This does not remove the constraint group, just the foundry override setting. The transient design route spec that was created by `set_foundry_override` is removed and the previous design route spec setting is restored.

Arguments

None

Value Returned

`true`	No constraint group is set as the foundry override.

Related Topics

set_foundry_override

get_foundry_override

unset_layer_constraint

```
unset_layer_constraint
     -constraint s_constraintName
     -layer s_layerName
     {-group s_groupName | -all}
     [-allMembers [true|false]]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-purpose s_purposeName]
     [-verbose [true|false]]
```

Description

Selectively removes a layer constraint or value for the given layer or layer purpose pair from a specific constraint group or from all constraint groups.

Arguments

`-all` Removes the constraint from all constraint groups.

`-allMembers [true|false]`

When set `true`, removes all members of the constraint from the given group (`-group`) or from all groups (`-all`). This is particularly useful for removing all members from an AND or OR group.

Default: `(false)` Removes only the first member of the constraint.

`-constraint` *`s_constraintName`*

Specifies the name of the constraint to remove.

`-group` *`s_groupName`* Specifies the group from which to remove the constraint.

`-hardness {hard | soft | preferred | default}`

Removes the constraint value for the given enforcement setting. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).

`default`	Removes the first value found: hard or soft.
`hard`	Removes the hard value.

	`preferred`	Removes the first value found: soft or hard.
	`soft`	Removes the soft value.
`-layer` *`s_layerName`*	Specifies the name of the layer associated with the constraint.	
`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`	Removes the constraint value for the given antenna model. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).	
`-purpose` *`s_purposeName`*	With `-layer`, specifies the layer purpose pair for the constraint.	
`-verbose [true|false]`	When set to `true`, the constraint name(s) and group(s) are output.	

Related Topics

set_layer_constraint

get_layer_constraint

unset_layerarray_constraint

```
unset_layerarray_constraint
     -constraint s_constraintName
     -layer_array {s_layerName…}
     {-group s_groupName | -all}
     [-allMembers [true|false]]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-verbose [true|false]]
```

Description

Selectively removes a layer array constraint value from a specific constraint group or from all constraint groups.

Arguments

`-all` Removes the constraint from all constraint groups.

`-allMembers [true|false]`

When set `true`, removes all members of the constraint from the given group (`-group`) or from all groups (`-all`). This is particularly useful for removing all members from an AND or OR group.

Default: `(false)` Removes only the first member of the constraint.

`-constraint` *`s_constraintName`*

Specifies the name of the constraint to remove.

`-group` *`s_groupName`* Specifies the group from which to remove the constraint.

`-hardness {hard | soft | preferred | default}`

Removes the constraint value for the given enforcement setting. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).

`default`	Removes the first value found: hard or soft.
`hard`	Removes the hard value.
`preferred`	Removes the first value found: soft or hard.

`soft`	Removes the soft value.
`-layer_array {`*`s_layerName`*`…}`	
	Specifies the layers associated with the constraint.
`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`	
	Removes the constraint value for the given antenna model. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).
`-verbose [true|false]`	When set to `true`, the constraint name(s) and group(s) are output.

Related Topics

set_layerarray_constraint

get_layerarray_constraint

unset_layerpair_constraint

```
unset_layerpair_constraint
     -constraint s_constraintName
     -layer1 s_layerName
     -layer2 s_layerName
     {-group s_groupName | -all}
     [-purpose1 s_purposeName]
     [-purpose2 s_purposeName]
     [-allMembers [true|false]]
     [-hardness {hard | soft | preferred | default}]
     [-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}]
     [-symmetric]
     [-verbose [true|false]]
```

Description

Selectively removes a layer pair constraint value for the given layers or layer purpose pairs from a specific constraint group or from all constraint groups.

Arguments

`-all` Removes the constraint from all constraint groups.

`-allMembers [true|false]`

When set `true`, removes all members of the constraint from the given group (`-group`) or from all groups (`-all`). This is particularly useful for removing all members from an AND or OR group.

Default: `(false)` Removes only the first member of the constraint.

`-constraint` *`s_constraintName`*

Specifies the name of the constraint to remove.

`-group` *`s_groupName`* Specifies the group from which to remove the constraint.

`-hardness {hard | soft | preferred | default}`

Removes the constraint value for the given enforcement setting. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).

`default` Removes the first value found: hard or soft.

	`hard`	Removes the hard value.
	`preferred`	Removes the first value found: soft or hard.
	`soft`	Removes the soft value.
`-layer1` *`s_layerName`*	Specifies the name of the first layer associated with the constraint to remove.	
`-layer2` *`s_layerName`*	Specifies the name of the second layer associated with the constraint.	
`-model {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`	Removes the constraint value for the given antenna model. If this was the last set value for the constraint, the constraint is removed from the constraint group(s).	
`-purpose1` *`s_purposeName`*	With `-layer1`, specifies the first layer purpose pair for the constraint.	
`-purpose2` *`s_purposeName`*	With `-layer2`, specifies the second layer purpose pair for the constraint.	
`-symmetric`	Indicates that `layer1` and `layer2` are considered interchangeable. By default, layer pair constraints are not symmetric.	
`-verbose [true|false]`	When set to `true`, the constraint name(s) and group(s) are output.	

Related Topics

set_layerpair_constraint

get_layerpair_constraint

unset_override_rulespec

```
unset_override_rulespec
```

Description

Unsets the override rulespec and disables the override rule lookups.

Arguments

None

Related Topics

get_override_rulespec

set_override_rulespec

Antenna Constraints

This topic lists the antenna constraints.

antenna	antennaMetalAreaFactor	antennaMetalCutBelowAreaFactor
antennaMetalDiodeAreaFactor	antennaMetalGateAreaFactor	antennaMetalMinusDiodeAreaFactor
antennaViaAreaFactor	antennaViaDiodeAreaFactor	antennaViaGateAreaFactor
cumulativeDAR	cumulativeGAR	cumulativeMetalAntenna
cumulativeSideWall	cumulativeViaAntenna	cumulativeViaDAR
cumulativeViaGAR	cumulativeViaSideWall	diodeAreaRatio
gateAreaRatio	maxFloatingArea	sideWall
validAntennaDiodes	validFillerCells	

Related Topics

Tcl Constraint Commands

Manage Constraints

antenna

Specifies the maximum antenna ratios for one or more antenna models (oxide thicknesses) for a layer. This constraint holds a value that specifies the maximum allowed ratio of the layer's antenna area to the gate area, and specifies how that maximum ratio changes if a diode is present on the wire.

antenna Quick Reference

Constraint Type	Layer
Value Types	`AntennaRatioValue`, `AntennaRatioValueArray`
Database Types	Technology
Scope	design, foundry
Category	Antenna

Value Types

`AntennaRatioValue` Specifies the default antenna model.

`AntennaRatioValueArray` Specifies more than one antenna model.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.

Related Topics

Antenna Constraints

antennaMetalAreaFactor

Specifies the multiplier for the metal node area calculation.

antennaMetalAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaMetalCutBelowAreaFactor

Specifies the multiplier for the via area connection up to the metal layer node calculation.

antennaMetalCutBelowAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaMetalDiodeAreaFactor

Specifies the multiplier for metal diode area calculation.

antennaMetalDiodeAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaMetalGateAreaFactor

Specifies the multiplier for metal gate area calculation.

antennaMetalGateAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaMetalMinusDiodeAreaFactor

Specifies the multiplier for the subtracted diode area calculation.

antennaMetalMinusDiodeAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaViaAreaFactor

Specifies the multiplier for via node area calculation.

antennaViaAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaViaDiodeAreaFactor

Specifies the multiplier for via diode area calculation.

antennaViaDiodeAreaFactor Quick Reference

Constraint Type	Simple
Value Types	DblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

DblValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

antennaViaGateAreaFactor

Specifies the multiplier for via gate area calculation.

antennaViaGateAreaFactor Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

FltValue	Specifies the multiplier to be used in the calculation.

Related Topics

Antenna Constraints

cumulativeDAR

Specifies the cumulative diode area ratio value taken from the `CumMetalAntenna` OpenAccess constraint.

cumulativeDAR Quick Reference

Constraint Type	Simple
Value Type	`LngDbl1DTblValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`LngDbl1DTblValue`	Specifies area values and corresponding ratio values.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeGAR

Specifies the cumulative gate area ratio taken from the `oacCumMetalAntenna` OpenAccess constraint.

cumulativeGAR Quick Reference

Constraint Type	Simple
Value Types	`DblValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`DblValue`	Specifies the cumulative gate area ratio.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeMetalAntenna

Specifies the maximum antenna ratios for one or more antenna models (oxide thicknesses) for area values accumulated through all the poly and metal layers attached to a gate. This constraint holds a value that specifies the maximum allowed ratio of this cumulative sum to the gate area and specifies how that maximum ratio changes if a diode is present on the wire.

cumulativeMetalAntenna Quick Reference

Constraint Type	Simple
Value Types	AntennaRatioValueArray, AntennaRatioValue
Database Types	Technology
Scope	design, foundry
Category	Antenna

Value Types

`AntennaRatioValue` Specifies the default antenna model.

`AntennaRatioValueArray`
Specifies more than one antenna model.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaCumRoutingPlusCut` (`BoolValue`) indicates whether the cumulative antenna ratio of the layer immediately below or the like layer immediately below for a layer's cumulative antenna calculation is included.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.

- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeSideWall

Specifies the cumulative side wall value taken from the `oacCumMetalAntenna` OpenAccess constraint.

cumulativeSideWall Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`BoolValue`	Indicates that the cumulative side wall value is taken from the `oacCumMetalAntenna` OpenAccess constraint.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeViaAntenna

Specifies the maximum antenna ratios for one or more antenna models (oxide thicknesses) for area values accumulated through all the cut and via layers attached to a gate. This constraint holds a value that specifies the maximum allowed ratio of this cumulative sum to the gate area, and specifies how that maximum ratio changes if a diode is present on the wire.

antenna Quick Reference

Constraint Type	Simple
Value Types	AntennaRatioValueArray, AntennaRatioValue
Database Types	Technology
Scope	design, foundry
Category	Antenna

Value Types

`AntennaRatioValue` Specifies the default antenna model.

`AntennaRatioValueArray`
Specifies more than one antenna model.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaCumRoutingPlusCut` (`BoolValue`) indicates whether the cumulative antenna ratio of the layer immediately below or the like layer immediately below for a layer's cumulative antenna calculation is included.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.

- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeViaDAR

Specifies the cumulative diode area ratio value taken from the `oacCumViaAntenna` OpenAccess constraint.

cumulativeViaDAR Quick Reference

Constraint Type	Simple
Value Types	`LngDbl1DTblValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`LngDbl1DTblValue`	Specifies area values and corresponding ratio values.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeViaGAR

Specifies the cumulative gate area ratio derived from the `oacCumViaAntenna` OpenAccess constraint.

cumulativeViaGAR Quick Reference

Constraint Type	Simple
Value Types	`DblValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`DblValue`	Specifies the cumulative gate area ratio.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

cumulativeViaSideWall

Specifies the cumulative side wall value taken from the `oacCumViaAntenna` OpenAccess constraint.

cumulativeViaSidewall Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`BoolValue`	Indicates that the cumulative side wall value is taken from the `oacCumViaAntenna` OpenAccess constraint.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffPMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

diodeAreaRatio

Specifies the diode area ratio value derived from the `oacAntenna` OpenAccess constraint.

diodeAreaRatio Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`FltValue`	Specifies the diode area ratio value.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffPMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

gateAreaRatio

Specifies the gate area ratio value derived from the OpenAccess `oacAntenna` constraint.

gateAreaRatio Quick Reference

Constraint Type	Layer
Value Types	`DblValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`DblValue`	Specifies the gate area ratio value.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaCumRoutingPlusCut` (`BoolValue`) indicates whether the cumulative antenna ratio of the layer immediately below or the like layer immediately below for a layer's cumulative antenna calculation is included.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

maxFloatingArea

Specifies the maximum area for floating metal shapes on a routing layer that are not connected to a diffusion or polysilicon gate. Similar to process antenna rules, maximum floating area rules apply only to the current layer and any lower layers (that is, all layers that have been fabricated up to the layer of interest). Maximum floating area rules can be used to avoid shorts between floating and non-floating metal wires that are caused by arcing due to charge build-up during processing steps. Applies only to routing layers.

Floating metal is defined as metal on the current layer that cannot trace a path to a diffusion connection or polysilicon gate, using only same layer or lower layer metal connections.

Grounded metal is defined as metal that can connect to a diffusion connection or polysilicon gat using only same layer or lower layer metal connections.

maxFloatingArea Quick Reference

Constraint Type	Simple
Value Types	`AreaValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Antenna

Value Type

`AreaValue`	Represents the maximum total area (in user units^2) for floating metal shapes on a routing layer that cannot trace a path to diffusion or polysilicon-gate using only same-layer or lower-layer metal connections.

Parameters

- `floatingAreaSpacing` (`LayerDualArrayTblValue`) specifies minimum spacing/ minimum parallel length pairs by layer. The `LayerDualArrayTblValue` syntax is:

  `{s_layerName i_numPairs {{f_minParSpacing f_minParallelLength}…}…}`

  where

s_layerName	Name of the layer.

i_numPairs Number of DualValue pairs for the layer.

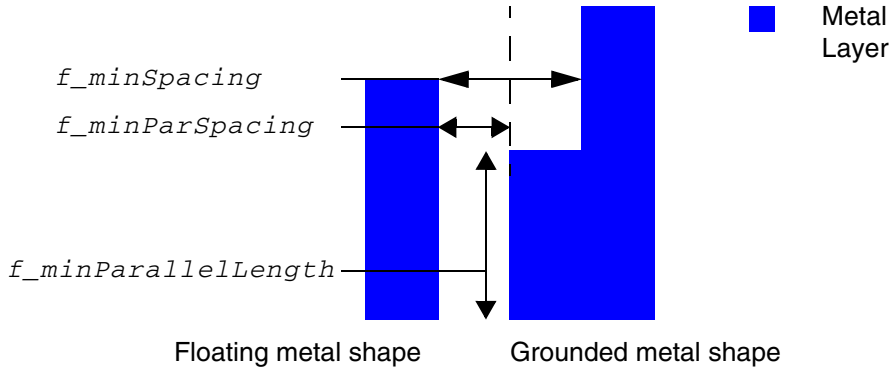
{*f_minParSpacing f_minParallelLength*}

The first DualValue pair for a layer must have the format

{*f_minSpacing* 0}

If the floating metal for the layer has area > `maxFloatingArea`, then the floating metal on the layer (and all connected lower layers when the `floatingMetal` parameter is `CONNECTED` or `ALLCONNECTED`) must be >= *f_minSpacing* away from all grounded metal on the layer or the rule fails.

For other DualValue pairs for the layer, floating metal that is >= *f_minParSpacing* from grounded metal must have >= *f_minParallelLength* at the *f_minSpacing* distance. *f_minSpacing* should be > *f_minParSpacing* values and if more than one DualValue pair of {*f_minParSpacing f_minParallelLength*} is given, then the *f_minParSpacing* values must be decreasing in value. You use the pair for the smallest *f_minParSpacing* that is >= the spacing between the floating metal and the grounded metal.



- `floatingMetal` (`StringAsIntValue`) specifies how the constraint is applied as described in the following table.

String	Integer Value	Constraint applies to:
`singleLayer`	0	Each individual floating metal shape on the routing layer must have an area <= `maxFloatingArea` or meet the minimum spacing to other grounded shapes.
`connected`	1	The area of floating metal shapes connected together on the layer.
`allConnected`	2	The area of floating metal shapes connected together on each layer defined by the constraint

Examples

Sets the required `floatingMetal` and `floatingAreaSpacing` parameters and the `maxFloatingArea` constraint.

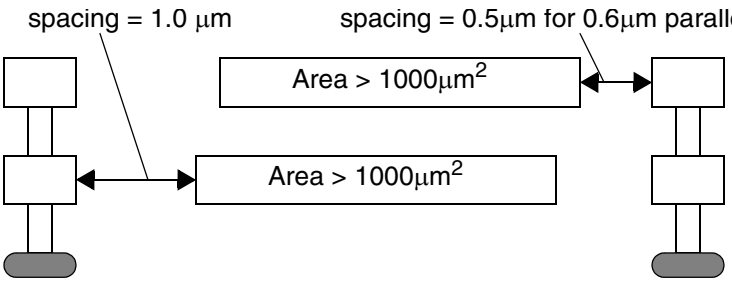
Format	Example

Tcl	`set_constraint_parameter -name floatingMetal \`
	`  -StringAsIntValue connected`
	`set_constraint_parameter -name floatingAreaSpacing \`
	`  -LayerDualArrayTblValue \`
	`  {Metal1 3  1.0 0.0  0.5 0.8  0.2 2.0  \`
	`   Metal2 2  0.6 0.0  0.3 0.9  \`
	`   Metal3 2  0.6 0.0  0.3 0.9  \`
	`   Metal4 2  0.6 0.0  0.3 0.9  \`
	`   Metal5 2  0.6 0.0  0.3 0.9  \`
	`   Metal6 2  0.6 0.0  0.3 0.9  }`
	`set_constraint -constraint maxFloatingArea -AreaValue 1000`

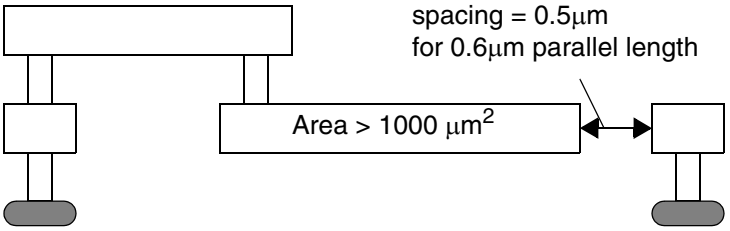
LEF	`LIBRARY LEF58_MAXFLOATINGAREA STRING`
	`"MAXFLOATINGAREA 1000 CONNECTED m1 m6`
	`  LAYERS m1 m1 SPACING 1.0 PARSPACING 0.5 0.8 0.2 2.0`
	`  LAYERS m2 m6 SPACING 0.6 PARSPACING 0.3 0.9 ;" ;`

The following figure shows the behavior when the constraint is applied for this example.

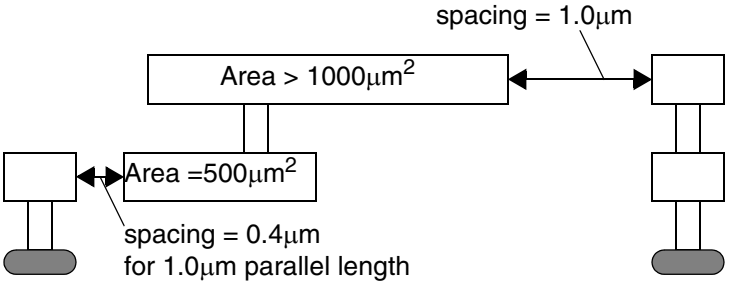
maxFloatingArea Example



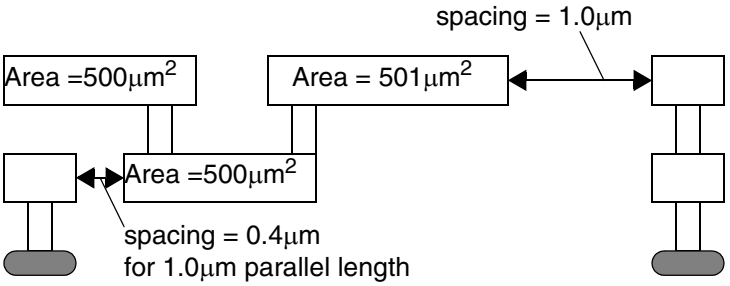
a) m2 Violation, m1 ok.
m2 does not meet spacing rule (0.5_{μ}m spacing requires 0.9_{μ}m parallel length).
m1 meets the 1.0μm spacing rule.



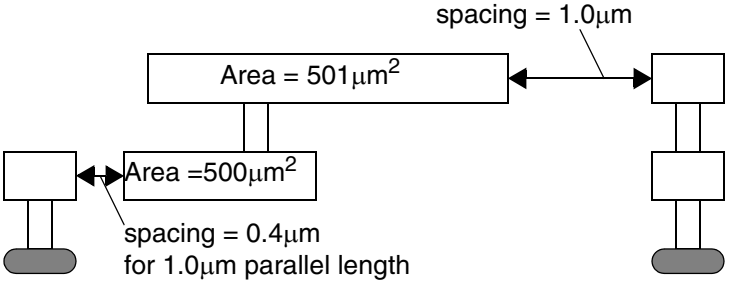
b) m1 Violation. m2 does not exist when m1 is created so m1 is still floating and does not meet the required 0.9μm parallel length for spacing >= 0.5μm.



c) m1 Violation. m2 spacing is ok but m2 floating area is connected to m1 floating area, so m2 must meet the required parallel length of 2.0μm for spacing >=0.2μm.



d) m1 Violation. m2 spacing is ok but total m2 floating area is > 1000μm2 connected to m1 floating area, so m1 must meet parallel length of 2.0μm for spacing >=0.2μm.



e) Ok. m1 area < 1000μm2 so no required spacing, and m2 area < 1000μm2 so no required spacing and m2 has no affect on m1 spacing. If ALL CONNECTED was specified, then this is a violation (m1+m2 area > 1000μm2).

Related Topics

Antenna Constraints

sideWall

Specifies whether the antenna constraints that are derived from the OpenAccess `oacAntenna` constraint represent the side area or the drawn (top surface) area.

sideWall Quick Reference

Constraint Type	Layer
Value Types	`BoolValue`
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

`BoolValue`	Specifies whether the antenna constraints represent the side area or the drawn top surface area.

Parameters

- `antennaAreaFactor` (`FltValue`) specifies a factor, between 0 and 1, for calculating the antenna area. The default value is 1.
- `antennaCumRoutingPlusCut` (`BoolValue`) indicates whether the cumulative antenna ratio of the layer immediately below or the like layer immediately below for a layer's cumulative antenna calculation is included.
- `antennaDiffAreaReduceFactor` (`Flt1DTblValue`) multiplies the conductor area by a floating point factor that depends on the diffusion area attached to the net. If the `antennaDiffMinusFactor` is non-zero, the subtraction of the diffusion area times the `antennaDiffMinusFactor` must occur after the metal and cut areas are multiplied by the `antennaDiffAreaReduceFactor`.
- `antennaDiffPlusFactor` (`FltValue`) determines how the diffusion area connected to the net is added to the area of the gate for antenna/gate ratio calculations. The default value is 0.0.
- `antennaDiffMinusFactor` (`FltValue`) determines how the diffusion area connected to the net is subtracted from the antenna area when calculating antenna/gate ratio for both metal and cut layers.

Related Topics

Antenna Constraints

validAntennaDiodes

Specifies the diode cells that can be used to fix antenna violations.

validAntennaDiodes Quick Reference

Constraint Type	Simple
Value Types	CellPathListValue
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

CellPathListValue	Lists the diodes that can be used to fix antenna violations.

Related Topics

Antenna Constraints

validFillerCells

Specifies the filler cells that can be used to fix antenna violations and place buffer cells.

validFillerCells Quick Reference

Constraint Type	Simple
Value Types	CellPathListValue
Database Types	not stored
Scope	design, foundry
Category	Antenna

Value Type

CellPathListValue	Lists the filler cells that can be used to fix antennas and place buffer cells.

Related Topics

Antenna Constraints

4

Area Constraints

This topic lists the area constraints:

- minArea
- minAreaEdgeLength
- minEnclosedArea
- oaMinRectArea

Related Topics

Tcl Constraint Commands

Manage Constraints

minArea

Specifies the minimum metal area for non-rectangular polygon shapes in user units^2.

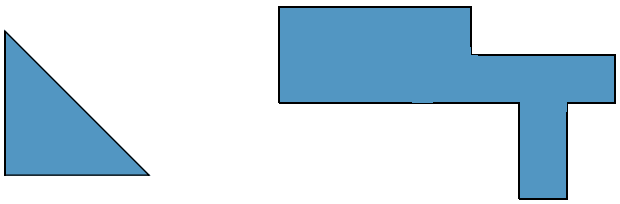
To specify the minimum area allowed for rectangular shapes on a layer, use the oaMinRectArea constraint.

minArea Quick Reference

Constraint Type	Layer
Value Types	AreaValue
Database Types	Design, Technology
Scope	design, foundry
Category	Area
Group Operators	AND, OR

Value Type

AreaValue	Specifies the minimum area in user units^2 for any shape on the layer.



The area of a shape on the specified layer must be greater than or equal to the `minArea` constraint value.

Examples

The following example, sets the minimum area for non-rectangular shapes on `Metal1` to `5.0` user units^2.

```
set_layer_constraint -constraint minArea -layer Metal1 -AreaValue 5.0
```

Related Topics

Area Constraints

minAreaEdgeLength

Specifies the minimum metal area for polygons on a layer that is larger than `minArea` and required when certain conditions exist:

- If all edges of the polygon are less than a given length
- If a rectangle of a given length and width cannot fit inside the polygon

For some processes, the area of a shape must be larger unless the shape has two adjacent edges, one of which is longer than a given length, and the other of shorter than a given length.

minAreaEdgeLength Quick Reference

Constraint Type	Layer
Value Types	`AreaValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Area
Group Operators	AND

Value Type

`AreaValue` — Specifies the minimum metal area (in user units^2) for polygons on the layer.

Optional Parameters

`shapeType` — Specifies the type of shapes to which the rule applies.

Type: `IntValue`
- 0 All shapes (default)
- 1 Rectangles only
- 2 Polygons/nonrectangular objects only

`exceptEdgeLength`	Specifies that the rule does not apply for polygons with at least one edge longer than this value in user units. This parameter cannot be specified with the `width` parameter for a layer. Type: `Value`
`exceptMinEdgeLength`	Specifies that the rule applies only if at least one edge has a length greater than or equal to this value in user units. Type: `Value`
`exceptMinSize`	Specifies that the rule applies only for shapes in which a minimum sized rectangle, given by the two values representing width and length, cannot fit. This parameter cannot be specified with the `width` or `exceptMinSizeArray` parameters for a layer. Type: `DualValue`
`exceptMinSizeArray`	Specifies that the rule applies for shapes in which a rectangle of any width/length pair, given by this parameter, cannot fit. This parameter cannot be used with `exceptMinSize`. Type: `LayerArrayValue`
`exceptStep`	Specifies that the rule does not apply for a polygon that has an edge with length greater than or equal to the first value and an adjacent edge of length less than the second value in user units. Type: `DualValue`
`width`	Specifies that the rule does not apply for any wire with all widths greater than or equal to this value, in user units. This parameter cannot be specified with the `exceptEdgeLength` or `exceptMinSize` parameters for a layer. Type: `Value`

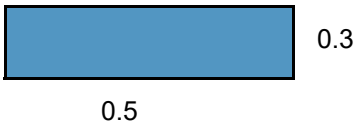
Examples

Example 1: minAreaEdgeLength with width Parameter

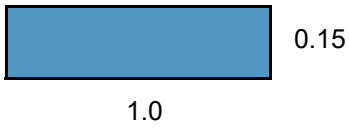
Requires that all polygons on `Metal1` must have a minimum area of 0.18 except when all widths for the wire are greater than or equal to 0.2, then its minimum area must be 0.1.

```
set_layer_constraint -constraint minArea -layer Metal1 -AreaValue 0.1
set_constraint_parameter -name width -Value 0.2
set_layer_constraint -constraint minAreaEdgeLength -layer Metal1 -AreaValue 0.18
```

Illustration of minAreaEdgeLength with width Parameter



a) PASS. Area = 0.3x0.5 = 0.15
Wire width = 0.3 > 0.2 so `minAreaEdgeLength` constraint does not apply and `minArea` constraint is satisfied because area = 0.15 > 0.1.



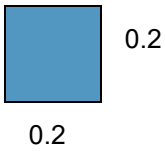
b) FAIL. Area = 0.15 x 1.0 = 0.15
Wire width = 0.15 < 0.2 so `minAreaEdgeLength` constraint applies but area = 0.15 < 0.18.

Example 2: minAreaEdgeLength with exceptEdgeLength

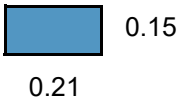
Requires that all polygons on `Metal1` must have a minimum area of 0.04, except if a polygon has at least one edge that is greater than or equal to 0.21, then its minimum area must be 0.03.

```
set_layer_constraint -constraint minArea -layer Metal1 -AreaValue 0.03
set_constraint_parameter -name exceptEdgeLength -Value 0.21
set_layer_constraint -constraint minAreaEdgeLength -layer Metal1 -AreaValue 0.04
```

Illustration of minAreaEdgeLength with exceptMinLength Parameter



a) PASS. Area = 0.2x0.2 = 0.04
No edge length is >= 0.21 so `minAreaEdgeLength` constraint of 0.04 applies and is met.



b) PASS. Area = 0.21x0.15 = 0.0315
At least one edge length is >= 0.21, so the `minAreaEdgeLength` of 0.04 does not apply. Minimum area of 0.03 is required for the `minArea` constraint and is met.

Example 3: minAreaEdgeLength with exceptMinSize

Requires that all polygons on `Metal1` must have a minimum area of 0.6, except if a rectangle of size 0.1 by 0.2 can fit inside the polygon, then the polygon's minimum area must be 0.5.

```
set_layer_constraint -constraint minArea -layer Metal1 -AreaValue 0.4
set_constraint_parameter -name exceptMinSize -DualValue {0.1 0.2}
set_layer_constraint -constraint minAreaEdgeLength -layer Metal1 -AreaValue 0.6
```

Illustration of minAreaEdgeLength with exceptMinSize Parameter

a) `minAreaEdgeLength` does not apply because a minimum sized rectangle fits inside.

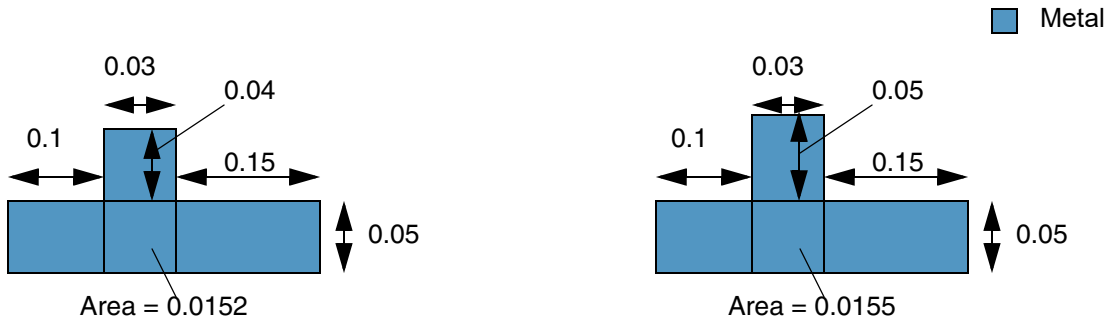
b) `minAreaEdgeLength` applies because a minimum sized rectangle cannot fit inside.

Example 4: minAreaEdgeLength with exceptStep

Requires that all polygons on `Metal1` must have a minimum area of 0.017, except when the polygon has an edge length greater than or equal to 0.12 with an adjacent edge that is less than 0.05, then it must have a minimum area of 0.015.

```
set_layer_constraint -constraint minArea -layer Metal1 -AreaValue 0.015
set_constraint_parameter -name exceptStep -DualValue {0.12 0.05}
set_layer_constraint -constraint minAreaEdgeLength -layer Metal1 -AreaValue 0.017
```

Illustration of minAreaEdgeLength with exceptStep Parameter



a) PASS. `minAreaEdgeLength` does not apply because the 0.15 edge (>= `exceptStep` first value of 0.12) has an adjacent edge of 0.04 (< `exceptStep` second value of 0.05). The 0.015 `minArea` is met.

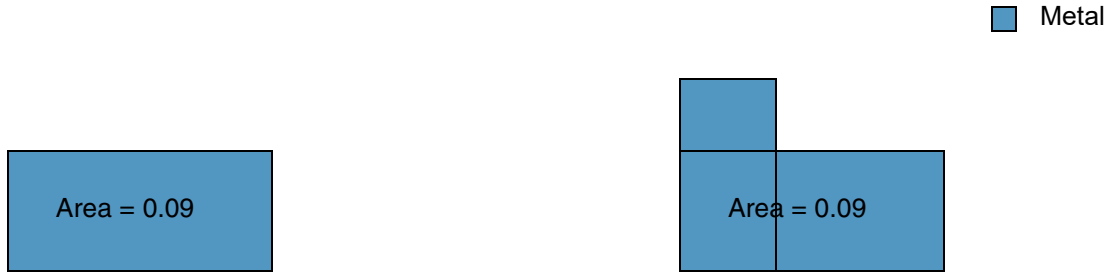
b) FAIL. There is no edge >= 0.12 having an adjacent edge < 0.05, so the 0.017 minimum area is required.

Example 5: minAreaEdgeLength with shapeType

Requires that all rectangles on `Metal1` must have a minimum area of 0.8, while any other polygons (non-rectangular objects) must have a minimum area of 0.10.

```
set_layer_constraint -constraint minArea -layer Metal1 -AreaValue 0.08
set_constraint_parameter -name shapeType -IntValue 2
set_layer_constraint -constraint minAreaEdgeLength -layer Metal1 -AreaValue 0.10
```

Illustration of minAreaEdgeLength with shapeType Parameter



a) PASS. A rectangle only requires a minimum area of 0.08 and is met.

b) FAIL. This is not a rectangle and minimum area must be >= 0.10 and is not met.

Related Topics

Area Constraints

minEnclosedArea

Specifies the minimum area for a hole, or an empty area, that is enclosed by metal. This constraint is often used in conjunction with the `minEnclosedSpacing` constraint for slotting.

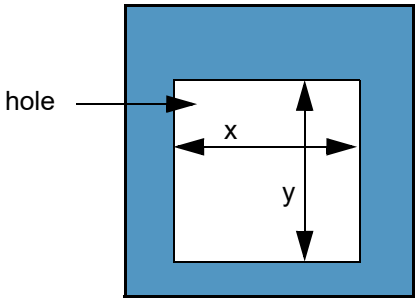
minEnclosedArea Quick Reference

Constraint Type	Layer
Value Types	`AreaValue`, `OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Area

Value Types

`AreaValue`

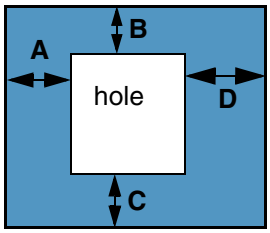
Specifies the minimum area allowed for a hole in the specified layer.



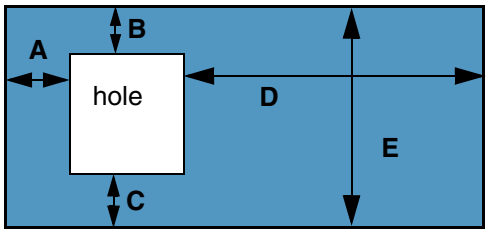
The area of a hole (x*y) in the specified layer must be greater than or equal to the `minEnclosedArea` constraint value.

`OneDTblValue`

Specifies the minimum hole area that is dependent on the effective width of the shapes surrounding the hole. The effective width is the smaller dimension of the surrounding rectangles.



a) The effective widths of the enclosing shapes are given by A, B, C, and D. The minimum hole area is determined by the largest of these widths.



b) The effective widths of the enclosing shapes are given by A, B, C, and E. The minimum hole area is determined by the largest width, E.

Examples

- Fixed Value

  Sets the minimum hole area on `Metal1` to `1.0` user unit.

  ```
  set_layer_constraint -constraint minEnclosedArea -layer Metal1 -AreaValue 1.0
  ```

- 1D Table: Index width

  Establishes a lookup table to set the minimum allowed area for a hole on `Metal1`, based on the width of the enclosing metal.

  For example, when the width of enclosing metal on `Metal1` is greater than or equal to `0.0005`, but less than `0.5605`, the minimum hole area is `0.12`; when the width is greater than or equal to `0.5605`, but less than `1.5005`, the minimum hole area is `0.18`.

  ```
  set_layer_constraint -constraint minEnclosedArea -layer Metal1 \
    -row_name width \
    -OneDTblValue {0.0005 0.12 0.5605 0.18 1.5005 0.5 3.0005  0.9}
  ```

Related Topics

Area Constraints

oaMinRectArea

Specifies the minimum area for rectangular shapes on a layer. Optionally, you can also specify a minimum and maximum width for the shape beyond which the constraint does not apply.

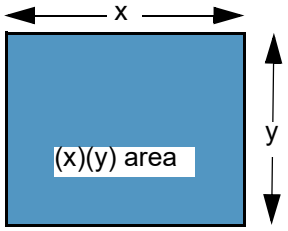
To specify the minimum area for non-rectangular, polygonal shapes, use the minArea constraint.

oaMinRectArea Quick Reference

Constraint Type	Layer
Value Types	`AreaValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Area

Value Type

`AreaValue` — Specifies the minimum area in user units^2 for rectangular shapes on the layer.



The area of a rectangle on the specified layer must be greater than or equal to the value specified by the `oaMinRectArea` constraint.

Optional Parameters

`width` — Specifies that the constraint applies only to shapes with width greater than or equal to this value.

Type: `Value`

`maxWidth`	Specifies that the constraint applies only to shapes with width less than or equal to this value. Type: `Value`

Examples

Example 1: oaMinRectArea

Sets the minimum rectangular area for a shape on Metal1 to 5.0 user units^2.

```
set_layer_constraint -constraint oaMinRectArea -layer Metal1 -AreaValue 5.0
```

Example 2: oaMinRectArea with width and maxWidth

Sets the minimum rectangular area for a shape on Metal1 to 0.08 user units^2 if the width of the shape is greater than or equal to 0.4 and less than or equal to 0.5.

```
set_constraint_parameter -name width -Value 0.4
set_constraint_parameter -name maxWidth -Value 0.5
set_layer_constraint -layer Metal1 -constraint oaMinRectArea -AreaValue 0.08
```

Related Topics

Area Constraints

5

Clearance Constraints

This topic lists the clearance constraints:

maxClearance	minCenterLineClearance	minClearance
minClearanceOverLayer	minClusterClearance	minCornerClearance
minCutRoutingClearance	minInnerVertexClearance	minNeighboringShapesClearance
minSameNetClearance	minSideClearance	minTouchingDirectionClearance
minVoltageClearance	shapeRequiredClearance	

Related Topics

Tcl Constraint Commands

Manage Constraints

maxClearance

Specifies the maximum spacing allowed between shapes on two different layers.

maxClearance Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`maxClearance` constraints have a `Value` that represents the maximum spacing, in user units, between shapes on two different layers.

Related Topics

Clearance Constraints

minCenterLineClearance

Specifies the required clearance between the centerline of a shape on one layer and a shape on a second layer.

minCenterLineClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

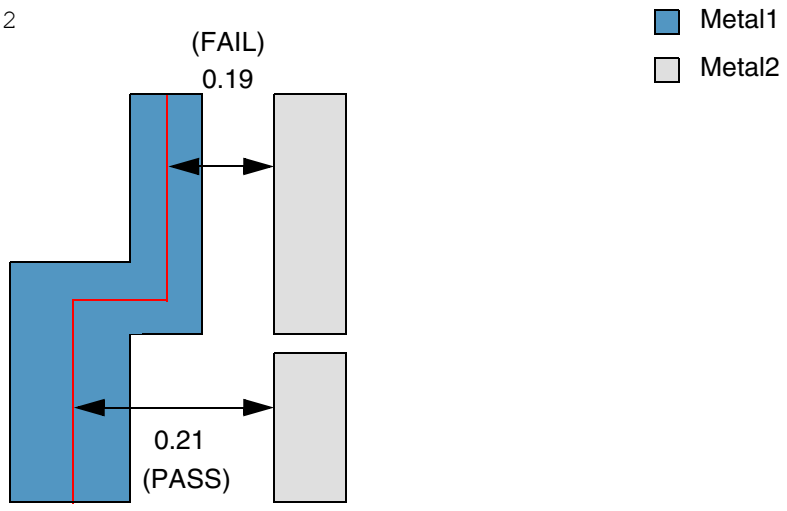
Value	Specifies the minimum required clearance in user units.

Examples

Specifies that the clearance between the `Metal2` shape and the centerline (in red) of the `Metal1` shape must be greater than or equal to 0.2 user units.

```
set_layerpair_constraint -constraint minCenterLineClearance \
  -layer1 Metal1 -layer2 Metal2 -Value 0.2
```

```
minCenterLineClearance    0.2
```



Related Topics

Clearance Constraints

minClearance

Sets the minimum spacing, edge-to-edge, between shapes on two different layers. This constraint applies only to non-intersecting shapes on the two different layers and ensures that shapes on the two layers do not cross.

This constraint is symmetric, implying that the minimum clearance between layer1 and layer2 is the same as the minimum clearance between layer2 and layer1.

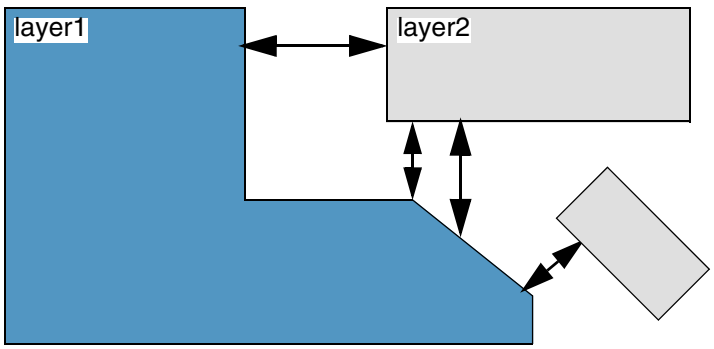
minClearance Quick Reference

Constraint Type	Layer pair (symmetric)
Value Types	Value, TwoDTblValue
Database Types	Design, Technology
Scope	net, route, Term, instTerm, net group, design, foundry
Category	Clearance

Value Types

Value

Specifies the minimum distance, in user units, between any shape on one layer and any shape, in every direction, on the other layer.



Any shape on either layer must be at least as far away, in every direction, from any shape on the other layer as the distance specified by the value of this constraint.

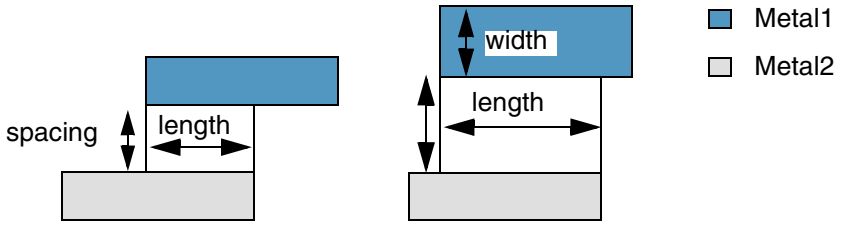
TwoDTblValue	Specifies the minimum clearance as a table of values based on `row_name` and `col_name` argument settings. For a detailed description of these argument settings, see `widthLengthTableType` parameter in the Optional Parameters section.

Optional Parameters

`widthLengthTableType`

Specifies the table type when `minClearance` is a TwoDTblValue. If this parameter is not specified, the `set_layerpair_constraint row_name` and `col_name` argument settings are used to determine the table type.

Type: IntValue
Value: 0
Width/Parallel run length table – The row lookup key represents the *width* of the wider of the two shapes, and the column lookup key represents the parallel run *length* between the two shapes. The table value represents the minimum required spacing.



`distanceMeasureType`

Specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1).

Type: IntValue
0 Euclidean
1 Manhattan

For information on this parameter, see Euclidean and Manhattan Spacing Constraints.

`coincidentAllowed`

When set `true`, then shapes can either meet the minimum clearance or their edges can be coincident. If `false` or not set, then the edges must meet the specified minimum clearance and cannot be coincident.

Type: `BoolValue`

`overlapNotAllowed`

When set to `true`, shapes are not allowed to overlap. By default, the checker allows same net shapes to overlap.

Type: `BoolValue`

`orthogonalSpacing`

Specifies the different-net clearance between objects on a cut layer and objects on a routing layer. This parameter can only be used with `minClearance` values of `Value`.

Type: `Value`

`oaSpacingDirection`

Specifies the measurement direction.

Type: `StringAsIntValue`
0 `anyDirection` (horizontally and vertically, default)
1 `horizontalDirection` (horizontally only)
2 `verticalDirection` (vertically only)

Examples

Example 1: minClearance Fixed Value

Sets the minimum clearance between shapes on `Metal1` and shapes on `Metal2` to `1.5`.

```
set_layerpair_constraint -constraint minClearance \
  -layer1 Metal1 -layer2 Metal2 -Value 1.5
```

Example 2: minClearance 2D Table: Indices width/length

In the following example, the minimum clearance is dependent on the parallel run length between a `Metal1` shape and a `Metal2` shape. To prevent crosstalk, more spacing is required for longer parallel run lengths. With the row width set to 0, the widths of the shapes are not considered.

- If the parallel run length is greater than or equal to 0.3 μm, then the minimum clearance is 0.2 μm.
- If the parallel run length is greater than or equal to `0.6` μm, then the minimum clearance is `0.4` μm.
- If the parallel length is greater than or equal to 1.0 μm, then the minimum clearance is `0.8` μm.

```
set_layerpair_constraint -constraint minClearance \
  -layer1 Metal1 -layer2 Metal2 \
  -hardness hard -row_name width -col_name length \
  -TblCols {           0.3  0.6  1.0 } \
  -TwoDTblValue    { 0 0.2  0.4  0.8 } \
  -row_interpolation snap_down \
  -row_extrapolation {snap_up snap_down} \
  -col_interpolation snap_down \
  -col_extrapolation {snap_up snap_down}
```

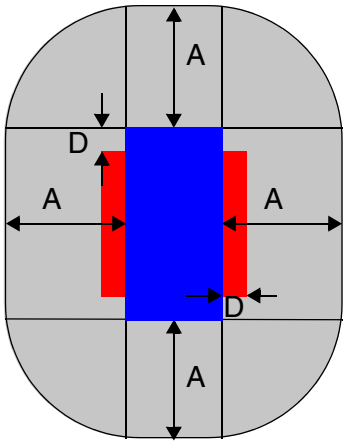
Example 3: minClearance with orthogonalSpacing

In the following example, the different-net spacing between objects on a cut layer and objects on a routing layer. The rule does not apply to same-net objects. The spacing check is done by using the following method:

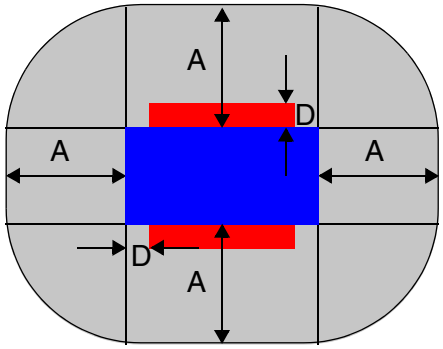
The difference between `minClearance` and `orthogonalSpacing` is calculated, and half of that difference is shrunk on two sides of the cut and grown on the other two sides. Then, a Euclidean spacing of the average of `minClearance` and `orthogonalSpacing` is applied to the newly formed rectangle geometry. The shrink and grow operations are done on the alternated sides to form another checking region. A violation occurs only when shapes on the routing layer are found in the gray regions of both check 1 and check 2.

```
set_constraint_parameter -name orthogonalSpacing -Value 0.2
set_layerpair_constraint -constraint minClearance \
  -layer1 "V1" -layer2 "Metal1" \
  -hardness hard -Value 0.1
```

Illustration of the Check Regions for minClearance with orthogonalSpacing



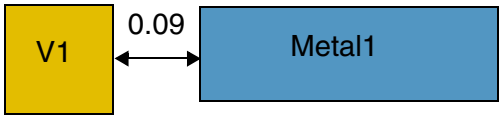
Check Region 1: Shrink horizontally, grow vertically



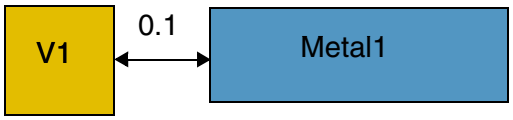
Check Region 2: Shrink vertically, grow horizontally

Start with a square red cut, shrink by D, `abs(`*`minClearance`* – *`orthogonalSpacing`*`)/2` on two sides of the cut and grow by D on the other two sides to result the blue rectangle. Then, spacing of A, (*`minClearance`* + *`orthogonalSpacing`*) / 2, is applied in a Euclidean fashion. A violation occurs only when shapes on the routing layer are found in the gray regions of both check region 1 and check region 2.

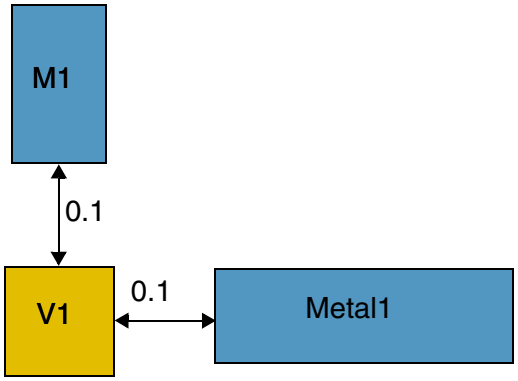
Illustration of minClearance with orthogonalSpacing



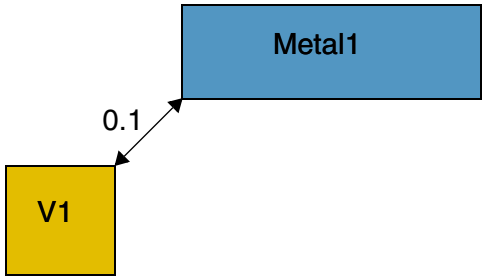
a) FAIL. Because 0.09 is less than the smallest spacing of 0.1.



b) PASS.Only one `Metal1` is less than 0.2 away on one side, but greater than or equal to 0.1.



c) FAIL. Both the sides have wires less than 0.2 distance away.



d) FAIL.The `Metal1` wire will be inside both checking regions.

Related Topics

Clearance Constraints

check_space

minClearanceOverLayer

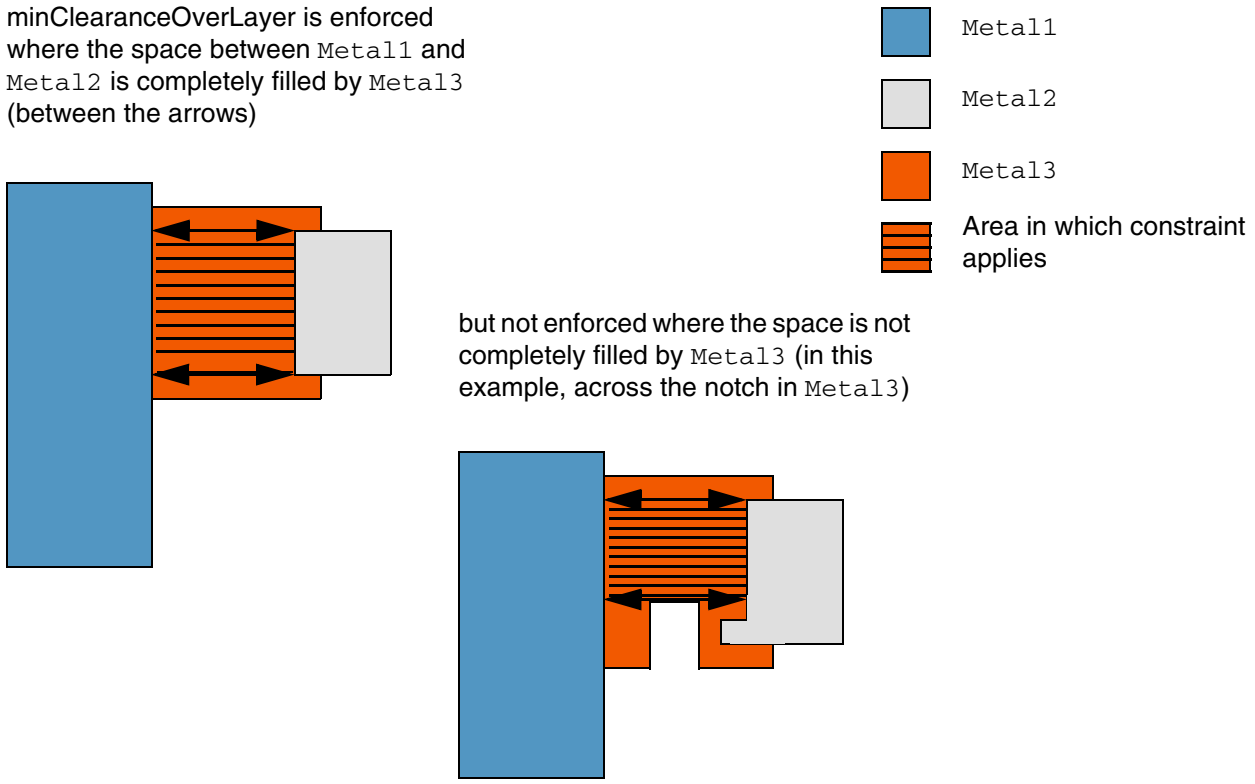
Sets the minimum clearance between layer1 and layer2 shapes where the space between the two is completely filled by layer3 shapes. For example, `minClearanceOverLayer` can be used to set the required clearance between implant (layer1) and gate (layer2) over diffusion (layer3), while `minClearance` specifies the minimum clearance between all other layer1 and layer2 shapes.

minClearanceOverLayer Quick Reference

Constraint Type	Layer array
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`Value`	Represents the minimum clearance in user units required between shapes on the two given layers wherever the space between them is completely filled by the third layer.



Examples

Sets the minimum clearance between `Metal1` and `Metal2` shapes to 2.0, wherever poly shapes completely fill any space between `Metal1` and `Metal2`.

```
set_layerarray_constraint -constraint minClearanceOverLayer \
  -layer_array {Metal1 Metal2 poly} -Value 2.0
```

Related Topics

Clearance Constraints

minClusterClearance

Specifies the minimum clearance between a group of aligned shapes and another shape on a different layer. The constraint specifies several conditions for the neighboring shapes to be considered as a valid group of shapes.

minClusterClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`Value`	Specifies the minimum clearance between the cluster and the other shape.

Required Parameters

The conditions specified by the following parameters must be satisfied for a cluster to be a valid cluster:

`shapeWidthRange`	Specifies that a valid cluster is only formed using shapes with widths in this range. Type: `RangeValue`
`numShapesRange`	Specifies that a valid cluster is only formed if the number of shapes in the cluster are in this range. Type: `RangeValue`
`spacingRange`	Specifies that a valid cluster is only formed if the neighboring shapes in a cluster are within this centerline spacing range. Type: `RangeValue`

Optional Parameters

`groupDirection`	Specifies that the groups can only be formed in the given direction. Type: `IntValue` 0 anyDirection (horizontally and vertically, default) 1 horizontalDirection (horizontally only) 2 verticalDirection (vertically only)
`centerLineDistance`	Specifies that the spacing measurement is done from the centerline of the cluster to the centerline of another shape. If this parameter is false, the spacing measurement is done from the edge of the cluster to the edge of another shape. **Note:** The intra-cluster spacing (specified by the parameter `spacingRange)` is always measured centerline. Type: `BoolValue`
`oaSpacingDirection`	Specifies the measurement direction. Type: `StringAsIntValue` anyDirection 0 Horizontally and vertically (default) horizontalDirection 1 Horizontally only verticalDirection 2 Vertically only
`otherWidthRange`	Specifies that the constraint applies if the width of the other shape is in this range. Type: `RangeArrayValue`
`parallelRunLength`	Specifies that the constraint applies only if the parallel run length between the cluster and the other shape is equal to or greater than this value. Type: `Value`

Examples

Example 1: minClusterClearance example

The minimum clearance between the cluster on `Metal1` and the other shape on `Metal2` must be greater than or equal to 3 user units under the following conditions:

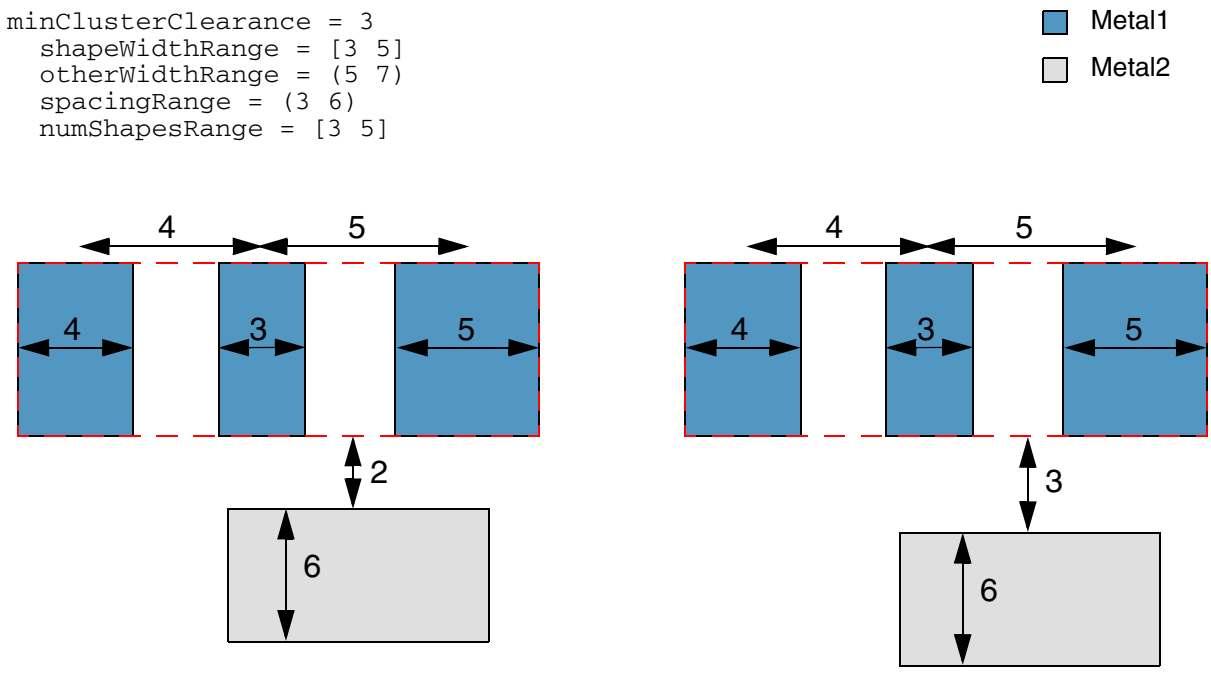
- The width of the shapes in the cluster on `Metal1` are greater than or equal to 3 and less than or equal to 5.

- The width of the other shape on `Metal2` is greater than 5 and less than 7.
- The centerline spacing between the shapes in the cluster is greater than 3 and less than 6.
- The number of shapes in the cluster is greater than or equal to 3 and less than or equal to 5.

```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[3 5]"}
set_layerpair_constraint -constraint minClusterClearance -Value 3 \
  -layer1 Metal1 -layer2 Metal2
```

Illustration for minClusterClearance

```
minClusterClearance = 3
  shapeWidthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [3 5]
```



a) FAIL. The spacing between the cluster and the `Metal2` shape is less than 3.

b) PASS. The spacing between the cluster and the Metal2 shape is greater than or equal to 3.

Example 2: minClusterClearance using centerLineDistance

```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
```

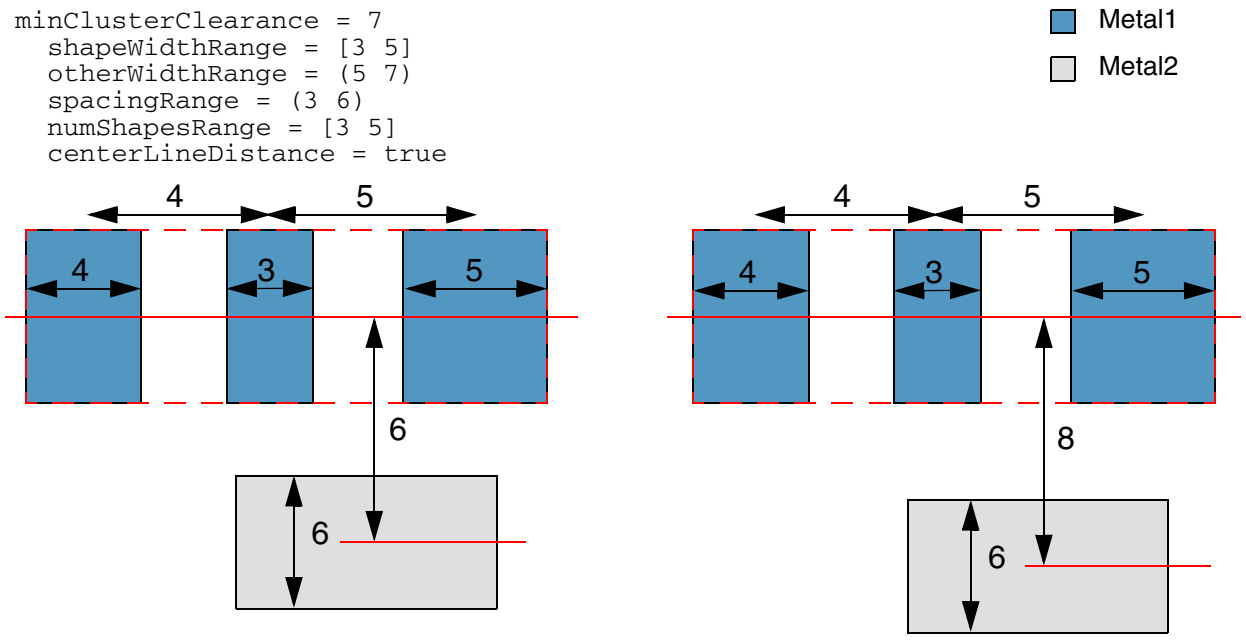
```
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[3 5]"}
set_constraint_parameter -name centerLineDistance -BoolValue true
set_layerpair_constraint -constraint minClusterClearance -Value 7 \
  -layer1 Metal1 -layer2 Metal2
```

The minimum clearance between the cluster on `Metal1` and the other shape on `Metal2` must be greater than or equal to 7 user units under the following conditions:

- Widths of shapes in the `Metal1` cluster are greater than or equal to and less than or equal to 5.
- Width of `Metal2` shape is greater than 5 and less than 7.
- Centerline spacing between the shapes in the cluster is greater than 3 and less than 6.
- Number of shapes in the cluster is greater than or equal to 3 and less than or equal to 5.
- Clearance measured between the centerline of the cluster and the centerline of another shape.

Illustration for minClusterClearance using centerLineDistance

```
minClusterClearance = 7
  shapeWidthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [3 5]
  centerLineDistance = true
```



a) FAIL. The spacing between the cluster and the `Metal2` shape is less than 7.

b) PASS. The spacing between the cluster and the shape on `Metal2` is greater than or equal to 7.

Example 3: minClusterClearance using groupDirection

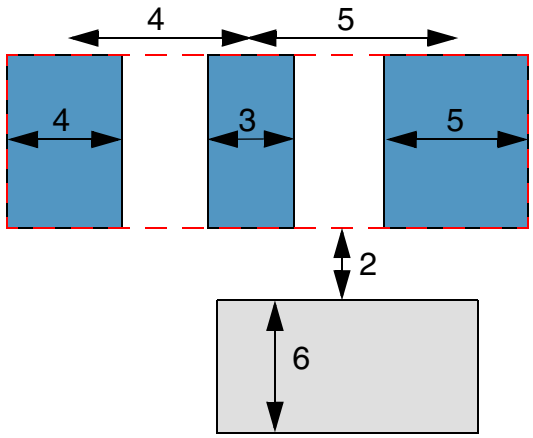
The minimum clearance between the cluster on `Metal1` and the other shape on `Metal2` must be greater than or equal to 5µm under the following conditions:

- Width of shapes in the `Metal1` cluster are greater than or equal to 3 and less than or equal to 5.
- Width of the other shape on `Metal2` is greater than 5 and less than 7.
- Centerline spacing between shapes in the cluster is greater than 3 and less than 6.
- Number of shapes in the cluster is greater than or equal to 3 and less than or equal to 5.
- The groups are formed in the vertical direction.

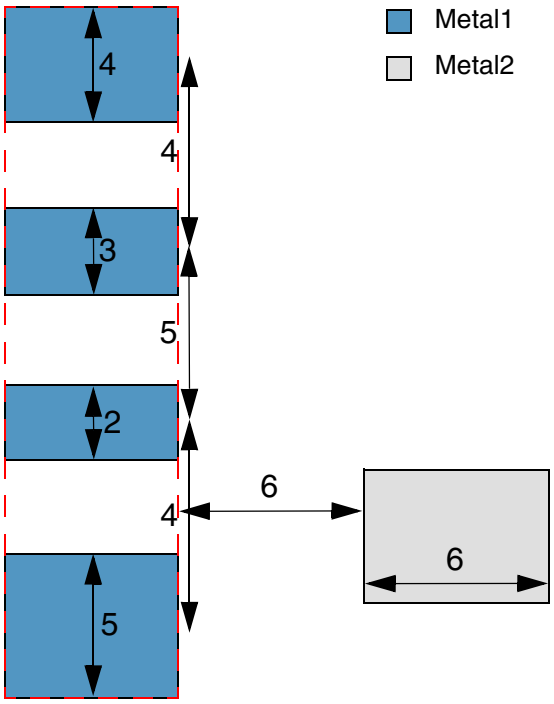
```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[3 5]"}
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name groupDirection -IntValue 2
set_layerpair_constraint -constraint minClusterClearance -Value 5 \
  -layer1 Metal1 -layer2 Metal2
```

Illustration for minClusterClearance using groupDirection

```
minClusterClearance = 5
  widthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [3 5]
  groupDirection = vertical
```



a) Constraint does not apply. The group direction is not vertical.



b) PASS. The spacing between the cluster and the other shape is greater than 5.

Related Topics

Clearance Constraints

check_layerpair_space

minClusterSpacing

minCornerClearance

Sets the minimum spacing required between the corners of shapes on different layers.

minCornerClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

Value	Specifies in user units the minimum spacing required between the corners of shapes on different layers.

Optional Parameters

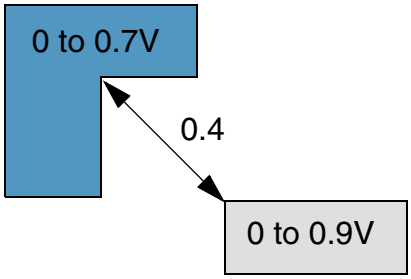
cornerType	Specifies the type of layer1 corners to which the constraint applies. Type: IntValue 0 allCorner 1 concaveCorner 2 convexCorner
otherCornerType	Specifies the type of layer2 corners to which the constraint applies. Type: IntValue 0 otherAllCorner 1 otherConcaveCorner 2 otherConvexCorner
deltaVoltage	Specifies that the constraint applies only if the maximum voltage difference between the two shapes is greater than this value. Type: FltValue

Examples

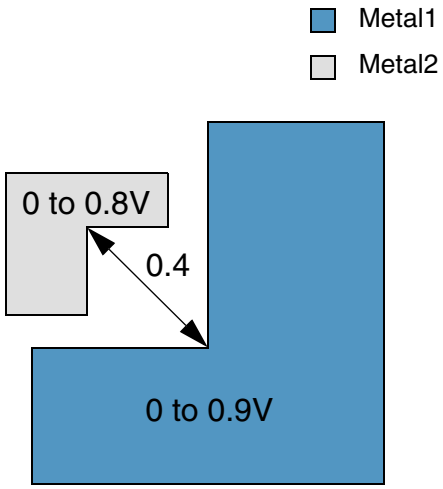
Sets a minimum spacing requirement of 0.5 user units between concave corners on `Metal1` and all corners on `Metal2` provided the maximum voltage difference between the shapes exceeds 0.8V.

```
set_constraint_parameter -name cornerType -IntValue 1
set_constraint_parameter -name otherCornerType -IntValue 0
set_constraint_parameter -name deltaVoltage -FltValue 0.8
set_layerpair_constraint -constraint minCornerClearance \
  -layer1 "Metal1" -layer2 "Metal2" \
  -Value 0.5
```

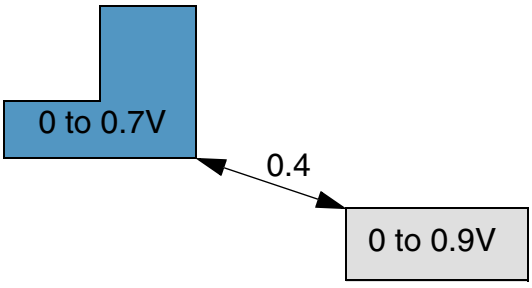
```
minCornerClearance      0.5
  cornerType            1 (concaveCorner)
  otherCornerType       0 (otherAllCorner)
  deltaVoltage          0.8
```



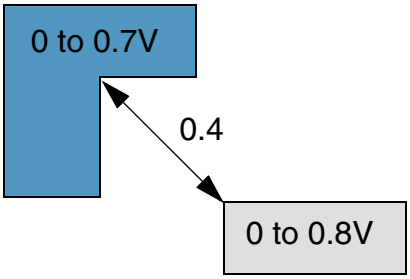
a) FAIL. The constraint applies between the `Metal1` concave corner and the `Metal2` convex corner with maximum voltage difference of 0.9 (greater than the specified `deltaVoltage` of 0.8), but the required spacing of 0.5 is not met.



b) FAIL. The constraint applies to the spacing between the `Metal1` concave corner and the `Metal2` concave corner with a maximum voltage different of 0.9, but the required spacing of 0.5 is not met.



c) Constraint does not apply to `Metal1` convex corners.



d) Constraint does not apply because the maximum voltage difference between the two shapes is 0.8.

Related Topics

Clearance Constraints

minCutRoutingClearance

Specifies the minimum clearance between specific cut classes and metal shapes. For rectangular cut classes, the clearance can optionally be enforced only between the short ends of the cut and metal shapes. Additionally, the clearance can optionally be between a cut shape and the concave corner of the wire that contains the cut.

minCutRoutingClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`Value`	Specifies the required clearance between the cut shapes on the first layer and the routing shapes on the second layer. The first layer must be the cut layer, and the second layer must be an interconnect or routing layer.

Optional Parameters

`cutClass`	Specifies the cut class width and length. Type: `DualValue`, in user units
`shortEdgeOnly`	Applies only when a rectangular cut class is specified by the `cutClass` parameter. If set to true, the clearance value applies only between the metal shape and the short edge of the rectangular cut shapes when there is a nonzero common parallel run length between them. This spacing requirement is ignored for shapes on the same net. Type: `BoolValue`

`exceptConnectivityType`

Specifies the vias for which this constraint does not apply, based on their connectivity. When `cutClass` is specified, this parameter should be set to `sameNetConnectivity`. When this parameter is not set, routing clearance applies in all cases.

Type: `IntValue`

`sameNetConnectivity` (same net)	1
`sameIslandConnectivity` (same metal)	2
`directConnectShapesConnectivity` (directly connected by the same metal)	3
`sameViaConnectivity` (same metal above and below)	4

`toConcaveCorner`

Specifies that the spacing between the cut shape and a concave corner of the metal shape in which the cut shape is placed must be greater than or equal to the constraint clearance value.

Type: `BoolValue`

`enclosingLayerWidth`

Specifies that the constraint applies if the shape that encloses the cut on `enclosingLayer` has width greater than or equal to this value.

Type: `Value`, in user units

`enclosingLayer`

Specifies that the constraint applies if the width of the shape that encloses the cut on `enclosingLayer` is greater than or equal to `enclosingLayerWidth`.

Type: `LayerValue`

`maskOverlap`

Requires that the minimum spacing be applied to the distance between cuts on layer1 and the overlap area between routing shapes on two different masks of layer2. The overlap area is created when shapes are stitched together. Cuts cannot touch or intersect the overlap area.

Type: `BoolValue`

`oaSpacingDirection`	Represents the direction in which the constraint applies. Type: `IntValue` 0 anyDirection (default) 1 horizontalDirection 2 verticalDirection
`anyOppositeExtension`	Specifies that the constraint applies only if a cut has an extension less than this value on any two opposite sides. Type: `Value`, in user units
`otherExtension`	Specifies that the constraint applies only if a cut has zero extension on two opposite sides and an extension less than this value on the other opposite sides. Type: `Value`, in user units
`parallelEdgeWithin`	Specifies that the constraint applies only if a cut with zero extension on two opposite edges has neighbors on both metal layers within this distance. Type: `Value`, in user units
`parallelExtension`	Specifies that the constraint applies only if the parallel cut edge has an extension that is less than this value. Type: `Value`, in user units
`parallelRunLength`	Specifies that the constraint applies only if a cut with zero extension on two opposite edges has neighbors on both metal layers, with parallel run greater than this value. Type: `Value`, in user units
`width`	Specifies that the constraint applies only when an edge of a concave corner has width less than or equal to the given value. Type: `Value`, in user units
`length`	Specifies that the constraint applies only when an orthogonal edge to the edge with width less than or equal to the given width has length greater than or equal to the given value. Type: `Value`, in user units

Examples

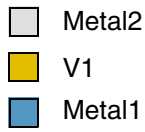
Example 1: minCutRoutingClearance with cutClass and shortEdgeOnly

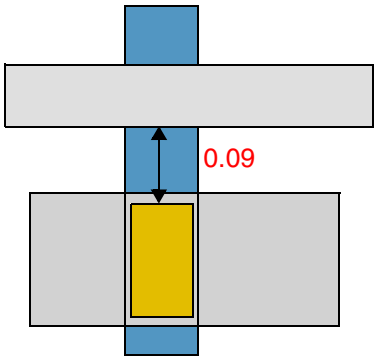
Sets the minimum clearance between the short end of 0.04 x 0.08 V1 cuts and `Metal2` shapes to 0.1 user units.

```
set_constraint_parameter -name cutClass -DualValue { 0.04 0.08 }
set_constraint_parameter -name shortEdgeOnly -BoolValue true
set_layerpair_constraint -constraint minCutRoutingClearance \
  -layer1 V1 -layer2 Metal2 -Value 0.1
```

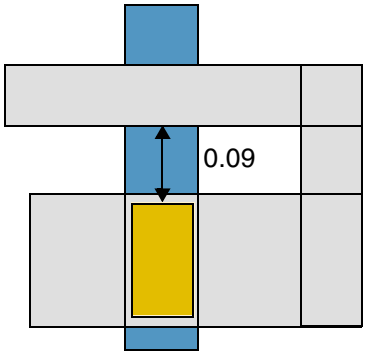
Illustration for minCutRoutingClearance with cutClass and shortEdgeOnly

```
minCutRoutingClearance V1 Metal2 0.1
   shortEdgeOnly  true
   cutClass       (0.04 0.08)
```





a) FAIL. The spacing between the short edge of the cut to an edge of a shape on `Metal2` is 0.09 (< 0.1).



b) Constraint does not apply because the shapes are on the same net.

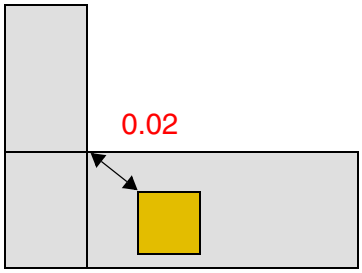
Example 2: minCutRoutingClearance with toConcaveCorner

Sets the minimum clearance between 0.02 x 0.02 V1 cuts and `Metal2` shapes to 0.03 user units. The spacing between 0.02 x 0.02 V1 cuts and concave corners of enclosing `Metal2` shapes must be greater than or equal to 0.03 user units.

```
set_constraint_parameter -name cutClass -DualValue {0.02 0.02}
set_constraint_parameter -name toConcaveCorner -BoolValue true
set_layerpair_constraint -constraint minCutRoutingClearance \
  -layer1 V1 -layer2 Metal2 -Value 0.03
```

Illustration for minCutRoutingClearance with toConcaveCorner

```
minCutRoutingClearance V1 Metal2 0.03
   toConcaveCorner   true
   cutClass          (0.02 0.02)
```



a) FAIL. The cut is at a distance of 0.02 (< 0.03) from the concave corner.

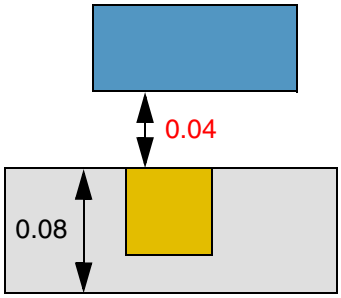
Example 3: minCutRoutingClearance with enclosingLayerWidth and enclosingLayer

Requires that the minimum clearance between cut class VA and metal shape `Metal1` must be greater than or equal to 0.5 user units, and the width of the shape that encloses the cut on the enclosing layer, `Metal2`, must be greater than or equal to 0.07 user units.

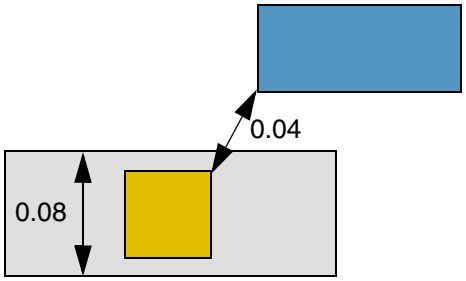
```
set_constraint_parameter -name className -StringValue VA
set_constraint_parameter -name cutClass -layer V1 \
  -DualValue {0.04 0.04}
set_constraint_parameter -name enclosingLayerWidth -Value 0.07
set_constraint_parameter -name enclosingLayer -LayerValue Metal2
set_layerpair_constraint -layer1 V1 -layer2 Metal1 \
  -constraint minCutRoutingClearance -Value 0.5
check_layerpair_space -lpp1 V1 -lpp2 Metal1 -diff_net -same_net
```

Illustration for minCutRoutingClearance with enclosingLayerWidth and enclosingLayer

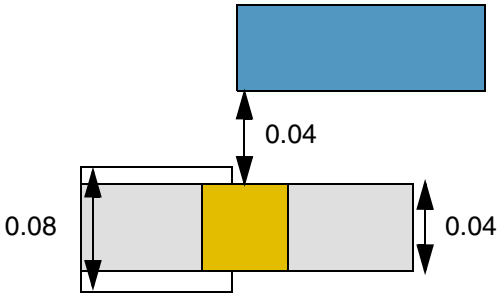
```
minCutRoutingClearance V1 Metal1 0.05
   enclosingLayer       Metal2
   enclosingLayerWidth  0.07
   cutClass V1          (0.04 0.04)
```



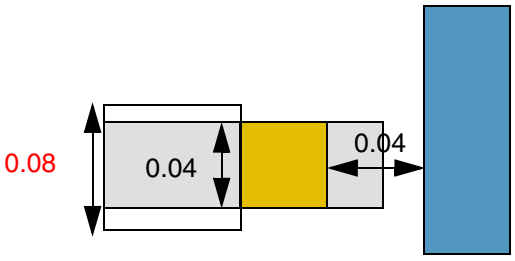
a) FAIL. Cut to metal spacing is 0.04 (<0.05).



b) FAIL. Cut to metal spacing is measured in Euclidean style.



c) FAIL. Part of the cut is inside a wider wire.



c) FAIL. The left edge of the cut touches a wire with width 0.08 (>0.07).

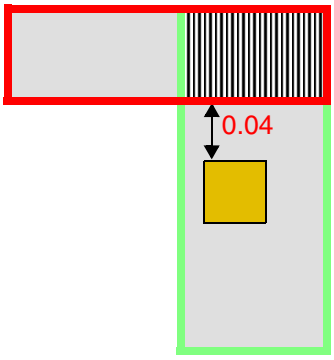
Example 4: minCutRoutingClearance with cutClass and maskOverlap

Sets a minimum clearance of 0.05 user units between a 0.06 x 0.06 V1 cut and the overlap area between `Metal2` shapes on mask1 and mask2.

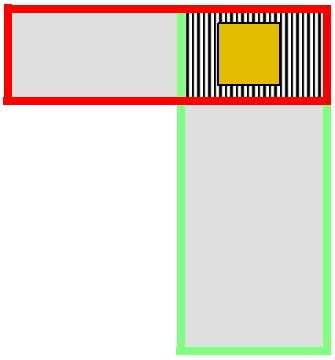
```
set_constraint_parameter -name cutClass -DualValue {0.06 0.06}
set_constraint_parameter -name maskOverlap -BoolValue true
set_layerpair_constraint -constraint minCutRoutingClearance \
  -layer1 V1 -layer2 Metal2 -Value 0.05
```

Illustration for minCutRoutingClearance with cutClass and maskOverlap

```
minCutRoutingClearance V1 Metal2 0.05
   maskOverlap  true
   cutClass     (0.06 0.06)
```



a) FAIL. The spacing between the V1 shape and the overlap area (between `Metal2` shapes on Mask1 and Mask2) is 0.04 (< 0.05).



b) FAIL. The V1 shape cannot touch or intersect the overlap area.

Related Topics

Clearance Constraints

check_layerpair_space

minInnerVertexClearance

Specifies the minimum spacing between the inside corner of a shape on `layer1` and a shape on `layer2`. The minimum spacing is satisfied when at least one of the `layer2` shape edges that are adjacent to the inside corner edges of the `layer1` shape meets the required spacing.

Optionally, this constraint does not apply if both edges of the `layer2` shape, which are adjacent to the `layer1` inside corner are at a distance greater than or equal to the specified distance.

minInnerVertexClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`minInnerVertexClearance` constraints have a `Value` that represents the minimum clearance.

Parameters

- `distance` (`Value`, optional) specifies that the constraint does not apply if both layer2 edges adjacent to the inside corner are greater than or equal to this distance from the layer1 shape that forms the inside corner.

Examples

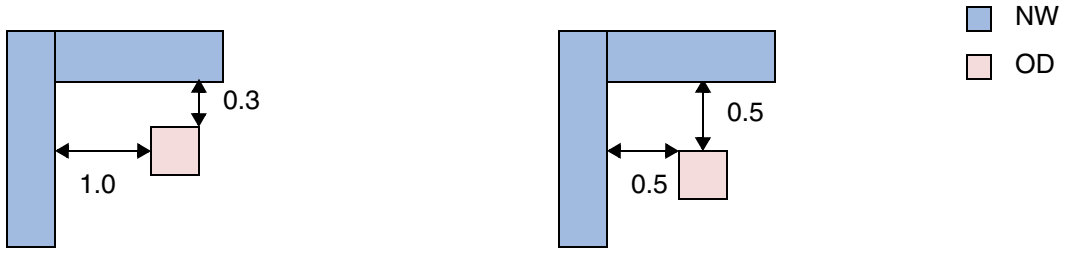
In this example, when NW forms an inside corner, then at least one adjacent OD edge must have a 1.0 clearance, or both OD edges must have clearance of 0.5 user units. All OD edges must have a clearance of at least a 0.3 user units from NW shapes.

Format	Example

Tcl	`set_layerpair_constraint -constraint minClearance \`
	`  -layer1 NW -layer2 OD -Value 0.3`
	
	`set_constraint_parameter -name distance -Value 0.5`
	`set_layerpair_constraint -constraint minInnerVertexClearance \`
	`  -layer1 NW -layer2 OD -Value 1.0`

Virtuoso	`spacings(`
	`  (minSpacing NW OD`
	`  0.3`
	`  )`
	
	`orderedSpacings(`
	`  (minInnerVertexSpacing NW OD`
	`  'distance 0.5`
	`  1.0`
	`  )`
	
	`)`

Illustration for minInnerVertexClearance



a) OK. Constraint applies and is met because one edge has a clearance of 1.0 and the other edge meets the minClearance of 0.3.

b) OK. Constraint applies and is met because both edges meet the clearance of 0.50.

c) Violation. At least one edge facing the inside corder must have a clearance of 1.0 user units.

d) Violation. One edge meets the minInnerVertexClearance of 1.0, but the other fails the minClearance of 0.3.

Related Topics

Clearance Constraints

check_space

minNeighboringShapesClearance

Specifies the minimum spacing between two shapes on different layers in the presence of neighboring shapes.

minNeighboringShapesClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

Value	Specifies the minimum spacing between layer1 and layer2 shapes in the presence of neighboring shapes.

Required Parameters

`widthRangeArray`

Specifies that the constraint applies only to the shapes whose widths are in this range.

Type: RangeArrayValue, in user units

`oppositeSpacingRangeArray`

Specifies that the distance between the opposite side of shape1 to the short side of the neighboring shape on layer2 fall within these ranges.

Type: RangeArrayValue, in user units

`lineEndSpacingRangeArray`

Specifies that the distance between the end-of-line of shape1 to the neighboring shape on layer2 is fall within these ranges.

Type: RangeArrayValue, in user units

Optional Parameters

`otherLayer`	Specifies the layer where the neighboring shapes are present. The default `otherLayer` is layer2. Type: `LayerValue`

Examples

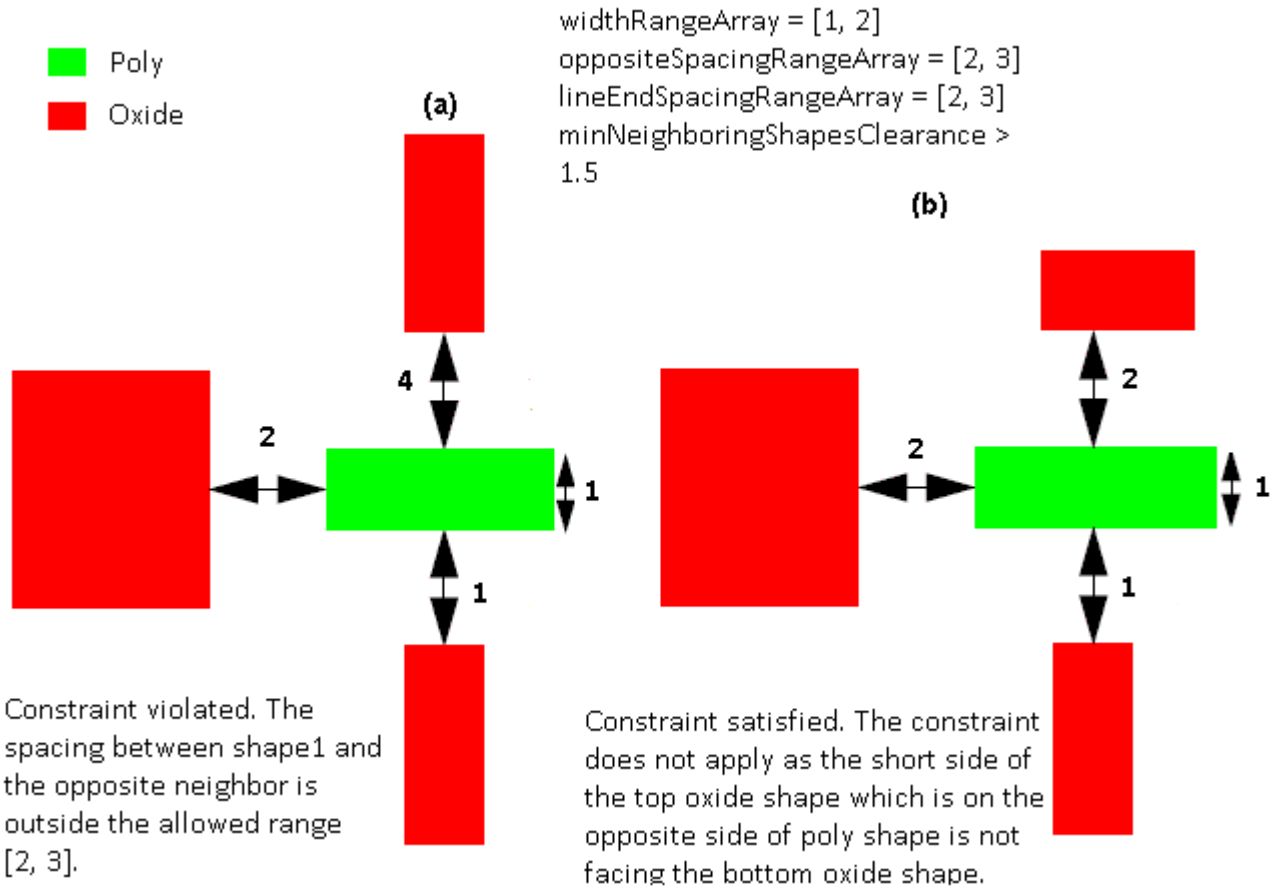
Example 1: minNeighboringShapesClearance with lineEndSpacingRangeArray

The minimum clearance between the shapes on `Poly` and `Oxide` layers, in the presence of neighboring shapes on the `Oxide` layer, must be greater than or equal to 1.5 user units under the following conditions:

- The width of the shape on the `Poly` layer is greater than or equal to 1 user unit and less than or equal to 2 user units.
- The distance between the opposite side of `Poly` shape to the short side of the neighboring shape on the `Oxide` layer is greater than or equal to 2 user units and less than or equal to 3 user units.
- The distance between the `Poly` shape to the neighboring shape on the end-of-line edge is greater than or equal to 2 user units and less than or equal to 3 user units.

```
set_constraint_parameter -name widthRangeArray -RangeArrayValue
{"[1 2]"}
set_constraint_parameter -name oppositeSpacingRangeArray
-RangeArrayValue {"[2 3]"}
set_constraint_parameter -name lineEndSpacingRangeArray
-RangeArrayValue {"[2 3]"}
set_layerpair_constraint -layer1 Poly -layer2 Oxide -symmetric false
-constraint minNeighboringShapesClearance -hardness hard
-Value 1.5
```

Illustration for minNeighboringShapesClearance with lineEndSpacingRangeArray



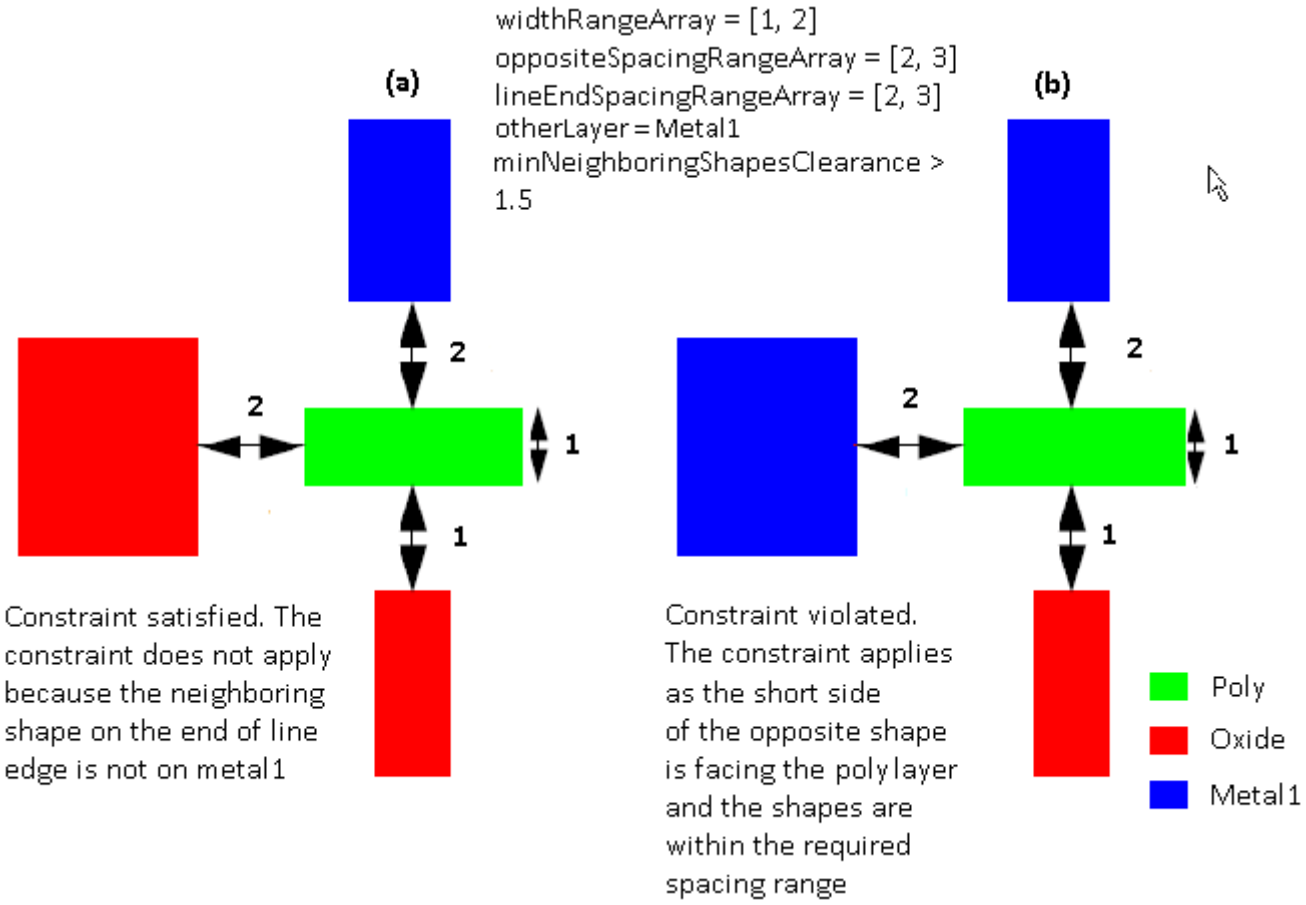
Example 2: minNeighboringShapesClearance with otherLayer

The minimum clearance between the shapes on the `Poly` and `Oxide` layers, in the presence of neighboring shapes on `Metal1`, must be greater than or equal to 1.5 user units under the following conditions:

- The width of the shape on the `Poly` layer is greater than or equal to 1 user unit and less than or equal to 2 user units.
- The distance between the opposite side of `Poly` shape to the short side of the neighboring shape on the `Oxide` layer is greater than or equal to 2 user units and less than or equal to 3 user units.
- The distance between the `Poly` shape to the neighboring shape on the end-of-line edge is greater than or equal to 2 user units and less than or equal to 3 user units.
- The neighboring shapes are present on `Metal1`.

```
set_constraint_parameter -name widthRangeArray -RangeArrayValue
{"[1 2]"}
set_constraint_parameter -name oppositeSpacingRangeArray
-RangeArrayValue {"[2 3]"}
set_constraint_parameter -name lineEndSpacingRangeArray
-RangeArrayValue {"[2 3]"}
set_constraint_parameter -name otherLayer -LayerValue "Metal1"
set_layerpair_constraint -layer1 Poly -layer2 Oxide -symmetric false
-constraint minNeighboringShapesClearance -hardness hard
-Value 1.5
```

Illustration for minNeighboringShapesClearance with otherLayer



Related Topics

Clearance Constraints

check_layerpair_space

minNeighboringShapesSpacing

minSameNetClearance

Sets the minimum spacing, edge-to-edge, between shapes on different layers but with the same net information.

This constraint is required if the same-net clearance is smaller than the different-net clearance specified with the minClearance constraint, or if vias on different layers have special stacking rules.

minSameNetClearance Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`minSameNetClearance` constraints have a `Value` that represents the minimum spacing, in user units, between shapes on layer1 and shapes on layer2 with the same net information.

Format	Example
Tcl	`set_layerpair_constraint -constraint minSameNetClearance \` `-layer1 s_layer1 -layer2 s_layer2 -Value f_space`

Virtuoso	`spacings(`
	`( minSameNetSpacing s_layer1 s_layer2 f_space )`
	`)`

Parameters

- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1).

Related Topics

Clearance Constraints

Euclidean and Manhattan Spacing Constraints

minSideClearance

Specifies the clearance between the shapes on different layers for a given (short or long) edge.

minSideClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Type

`minSideClearance` constraint has the following value type:

- Value specifies the minimum clearance between the edges of the two shapes.

Parameters

- `spacingType` (IntValue, required) specifies the type of edges of both the shapes.

spacingType Values

spacingType	Integer Value
anySideToAnySide	0
shortSideToShortSide	1
shortSideToLongSide	2
shortSideToAnySide	3
longSideToLongSide	4
longSideToAnySide	5

- `parallelRunLength` (`Value`, optional) specifies that the constraint applies only if the parallel run length between the edges of the shapes is equal to or greater than this value.

  Note: This constraint supports negative parallel run length.

- `widthRangeArray` (`RangeArrayValue`, optional) specifies that the constraint applies if width of the layer1 shape is in this range.

- `otherWidthRangeArray` (`RangeArrayValue`, optional) specifies that the constraint applies if width of the layer2 shape is in this range.

Examples

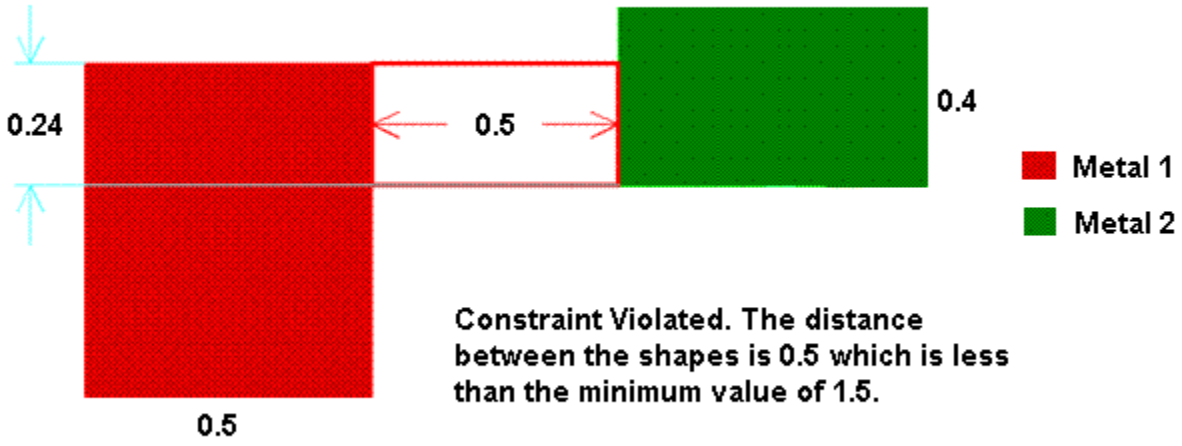
In this example, the minimum clearance between `Metal1` and `Metal2` shapes must be greater than or equal to 1.5µm under the following conditions:

- The spacing is measured from any side of the `Metal1` shape to any side of the `Metal2` shape.
- The parallel run length between the two shapes is greater than or equal to 0.1µm.
- The width of the `Metal1` shape is greater than 0.2µm and less than 0.9µm.
- The width of the `Metal2` shape is greater than 0.2µm and less than 0.9 µm.

Format	Example

Tcl	`set_constraint_parameter -name spacingType -IntValue 0`
	`set_constraint_parameter -name parallelRunLength -Value 0.1`
	`set_constraint_parameter -name widthRangeArray -RangeArrayValue { "(0.2 0.9)" }`
	`set_constraint_parameter -name otherWidthRangeArray -RangeArrayValue { "(0.2 0.9)" }`
	`set_layerpair_constraint -constraint minSideClearance -layer1 Metal1 -layer2 Metal2 -Value 1.5`
	`check_layerpair_space -same_net -diff_net -lpp1 Metal1 -lpp2 Metal2`

Illustration for minSideClearance Using Value



Related Topics

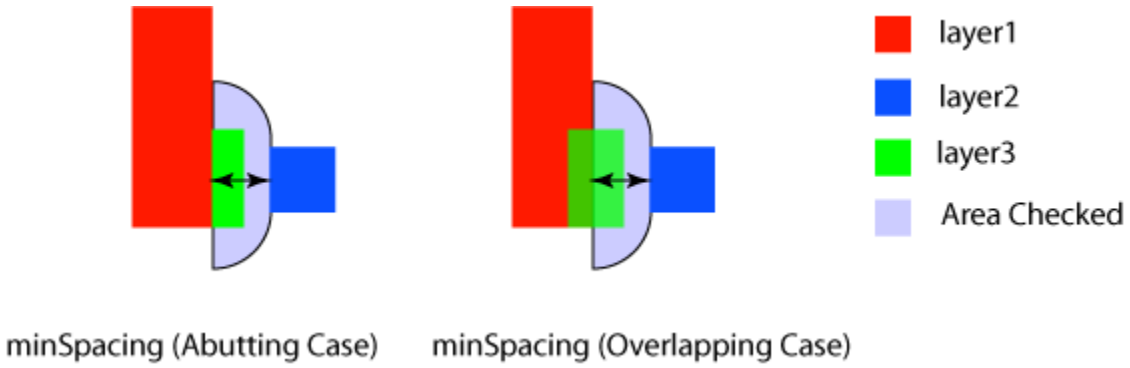
Clearance Constraints

minSideSpacing

check_layerpair_space

minTouchingDirectionClearance

Specifies the minimum required clearance between layer1 and layer2 in the direction normal to the layer1 edge when that edge is touched by a third layer. Touching is defined as either abutting of the third layer with layer1 or overlapping of layer1 by the third layer. The clearance is measured between the layer1 edge that touches the third layer and adjacent layer2 shapes.



Use this constraint to specify the clearance between poly and gate in the direction of the channel.

minTouchingDirectionClearance Quick Reference

Constraint Type	Layer array
Value Types	`Value`, `DualValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Types

`minTouchingDirectionClearance` constraints have a

- `Value` specifies the clearance in user units.
- `DualValue` specifies the clearance when the spacing measurement method is set to Manhattan. The first number of the pair describes the required clearance in the direction normal to the touching edge and the second number describes the required clearance parallel to the touching edge.

Parameters

- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as either Euclidean (`0`, default) or Manhattan (`1`).

Related Topics

Clearance Constraints

Euclidean and Manhattan Spacing Constraints

minVoltageClearance

Specifies the minimum spacing required for wires of different voltages in the given routing layer to all cut via shapes in the adjacent cut layers above and below.

minVoltageClearance Quick Reference

Constraint Type	Layer pair
Value Types	`FltHeader1DTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Types

`minVoltageClearance` constraints have a `FltHeader1DTblValue` that consists of `FltValue`-`Value` pairs where `FltValue` represents the voltage and `Value` is the minimum clearance, in user units.

To determine the minimum clearance, the maximum voltage swing between shapes is calculated as (`max`(max voltage of each shape) - (`min`(min voltage of each shape))). This voltage is compared with the voltage values in the table to find the first value that is less than or equal to the calculated voltage swing. The corresponding minimum clearance is required.

`minVoltageClearance` constraints can be applied to layer purpose pairs and nets. For more information on these applications, see Voltage-Dependent Rule Support.

Examples

For `Metal1` and `V1` shapes in this example,

- For voltage swings >= 0.0 and < 2.5, the minimum clearance is 0.1.
- For voltage swings >= 2.5 and < 6.0, the minimum clearance is 0.22.
- For voltage swings >= 6.0, the minimum clearance is 0.35.

For `Metal2` and `V1` shapes in this example,

- For voltage swings >= 0.0 and < 2.5, the minimum clearance is 0.1.

- For voltage swings >= 2.5 and < 6.0, the minimum clearance is 0.23.
- For voltage swings >= 6.0, the minimum clearance is 0.36.

Format	Example

Tcl	`set_layerpair_constraint -constraint minVoltageClearance \`
	` -layer1 Metal1 -layer2 V1 -symmetric true -hardness hard \`
	` -row_name voltage \`
	` -FltHeader1DTblValue { 0.0 0.1  2.5 0.22  6.0 0.35 } \`
	` -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down}`
	`set_layerpair_constraint -constraint minVoltageClearance \`
	` -layer1 Metal2 -layer2 V1 -symmetric true -hardness hard \`
	` -row_name voltage \`
	` -FltHeader1DTblValue { 0.0 0.1  2.5 0.23  6.0 0.36 } \`
	` -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down}`

LEF	`LAYER Metal1`
	`PROPERTY LEF58_VOLTAGESPACING`
	`  "VOLTAGESPACING TOCUT ABOVE`
	`   0.0 0.1`
	`   2.5 0.22`
	`   6.0 0.35 ;"`
	`LAYER Metal2`
	`PROPERTY LEF58_VOLTAGESPACING`
	`  "VOLTAGESPACING TOCUT BELOW`
	`   0.0 0.1`
	`   2.5 0.23`
	`   6.0 0.36 ;"`

Virtuoso	`spacingTables(`
	`  ( minVoltageSpacing "Metal1" "V1"`
	`   ( "voltage" nil nil )`
	`   ( 0.0 0.1`
	`     2.5 0.22`
	`     6.0 0.35`
	`   )`
	`  )`
	`  ( minVoltageSpacing "Metal2" "V1"`
	`   ( "voltage" nil nil )`
	`   ( 0.0 0.1`
	`     2.5 0.23`
	`     6.0 0.36`
	`   )`
	`  )`
	`);spacingTables`

Related Topics

Clearance Constraints

shapeRequiredClearance

Requires the existence of another shape on a different layer. The direction of measurement of the other shape must be either vertical or horizontal and must be present for the full parallel run length.

shapeRequiredClearance Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	RangeArrayValue, RangeArray1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Clearance

Value Types

`shapeRequiredClearance` constraint has the following value types:

- `RangeArrayValue` specifies that there must be at least one layer2 shape within one of these ranges from layer1.
- `RangeArray1DTblValue` specifies the ranges within which at least one layer2 shape must exist for a layer1 shape of a given width.

Parameters

- `oaSpacingDirection` (`StringAsIntValue`, required) specifies the measurement direction in which the distance to the layer2 shape is measured
  anyDirection 0 Horizontally and vertically (default)
  horizontalDirection 1 Horizontally only
  verticalDirection 2 Vertically only
- `twoSides` (`BoolValue`, optional) if true, specifies that the layer2 shape is required on both the sides. The default value of this parameter is false.
- `widthRangeArray` (`RangeArrayValue`, optional) specifies that the constraint only applies to those shapes on Layer1 whose widths are in this range.
- `sideExtension` (`Value`, optional) specifies the extent to which layer1 should be extended orthogonally to measurement direction before measuring its distance from layer2.

Examples

Example 1 — shapeRequiredClearance using twoSides

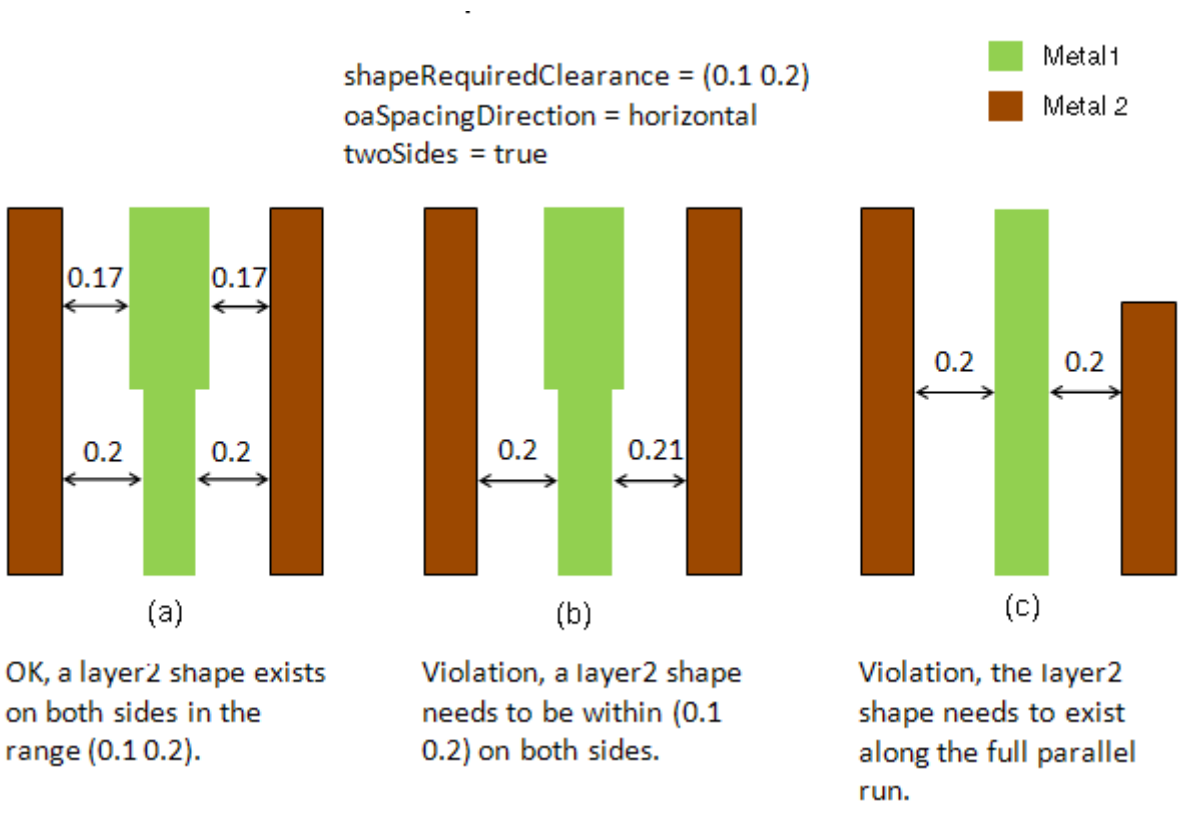
In this example, the clearance between the shape on `Metal1` and the required shape on `Metal2` must be greater than 0.1µm and less than 0.2µm under the following conditions:

- The distance to the `Metal2` shape is measured in the horizontal direction.
- A `Metal2` shape is required on both sides.

Format	Example

Tcl	`set_constraint_parameter -name oaSpacingDirection -StringAsIntValue horizontalDirection`
	`set_constraint_parameter -name twoSides -BoolValue true`
	`set_layerpair_constraint -layer1 Metal1 -layer2 Metal2 -constraint shapeRequiredClearance -RangeArrayValue {"(0.1 0.2)"}`

Illustration for shapeRequiredClearance using twoSides



Example 2 — shapeRequiredClearance using sideExtension

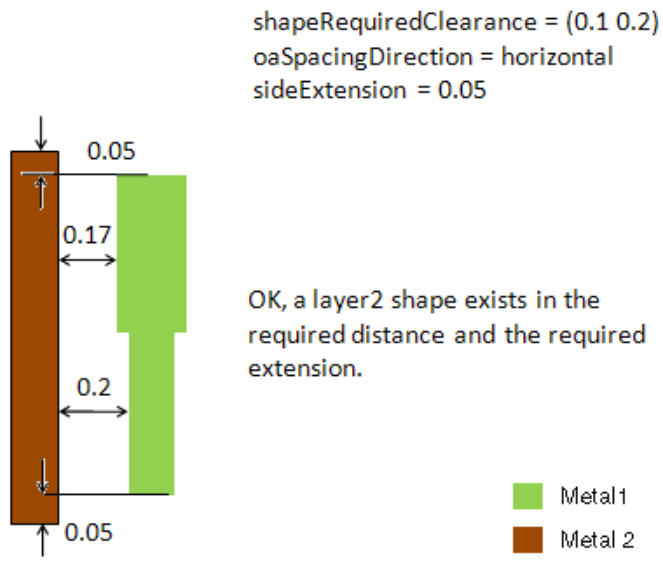
In this example, the clearance between the shape on `Metal1` and the required shape on `Metal2` must be greater than 0.1µm and less than 0.2µm under the following conditions:

- The distance to the `Metal2` shape is measured in the horizontal direction.
- The `Metal1` is extended orthogonally by 0.05 in the vertical direction before measuring its distance from `Metal2`.

Format	Example

Tcl	`set_constraint_parameter -name oaSpacingDirection -StringAsIntValue horizontalDirection`
	`set_constraint_parameter -name sideExtension -Value 0.05`
	`set_layerpair_constraint -layer1 Metal1 -layer2 Metal2 -constraint shapeRequiredClearance -RangeArrayValue {"(0.1 0.2)"}`

Illustration for shapeRequiredClearance using sideExtension



Related Topics

Clearance Constraints

shapeRequiredSpacing

check_layerpair_space

6

CMP Constraints

This topic lists the CMP constraints:

CMPAnalysisGrid	CMPRegionHalo
maxDensityGradient	maxOxideCopperStepHeight
maxSurfaceHeightRange	minThicknessGradient
maxThicknessGradientPercent	maxThicknessThreshold
maxThicknessThresholdPercent	minDensityGradient
minThicknessGradient	minThicknessGradientPercent
minThicknessThreshold	minThicknessThresholdPercent

Related Topics

Tcl Constraint Commands

Manage Constraints

CMPAnalysisGrid

Specifies, in user units, the size of the square grid to use for geometric extraction, CMP analysis, and color maps.

CMPAnalysisGrid Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`CMPAnalysisGrid` constraints have a `Value` that represents the size of the grid, in user units. The default is 20 (20x20 grid).

Examples

The following example sets the grid to 50 user units.

```
set_constraint -constraint CMPAnalysisGrid -Value 50
```

Related Topics

CMP Constraints

CMPRegionHalo

Specifies in user units the amount by which the analysis region should be bloated before region-based CMP analysis is run, in order to achieve reasonably accurate results.

CMPRegionHalo Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`CMPRegionHalo` constraints have a `Value` that represents the expansion size, in user units.

Examples

The following example sets the bloat size to 1.0 μm.

```
set_constraint -constraint CMPRegionHalo -Value 1.0
```

Related Topics

CMP Constraints

maxDensityGradient

Specifies the maximum metal density gradient (delta), as a percentage, between the current window and its neighbors.

maxDensityGradient Quick Reference

Constraint Type	Layer
Value Types	FltValue
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxDensityGradient` constraints have a FltValue representing the maximum metal density gradient (delta), as a percentage, between the current window and its neighbors.

Related Topics

CMP Constraints

maxOxideCopperStepHeight

Specifies the maximum difference between oxide height and copper thickness, in angstroms, for CMP analysis.

maxOxideCopperStepHeight Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxOxideCopperStepHeight` constraints have a `FltValue` that represents the maximum difference between oxide height and copper thickness, in angstroms.

Related Topics

CMP Constraints

maxSurfaceHeightRange

Specifies the maximum range of copper and/or oxide topology, in angstroms, for CMP analysis.

maxSurfaceHeightRange Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxSurfaceHeightRange` constraints have a `FltValue` that specifies the maximum range of copper and/or oxide topology, in angstroms.

Related Topics

CMP Constraints

maxThicknessGradient

Specifies the maximum copper thickness gradient (delta), as an absolute difference in angstroms, between the current window and its neighbors.

maxThicknessGradient Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxThicknessGradient` constraints have a `FltValue` representing the maximum copper thickness gradient (delta), as an absolute difference in angstroms, between the current window and its neighbors.

Related Topics

CMP Constraints

maxThicknessGradientPercent

Specifies the maximum copper thickness gradient (delta), as a percentage, between the current window and its neighbors.

maxThicknessGradientPercent Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxThicknessGradientPercent` constraints have a `FltValue` representing the maximum copper thickness gradient (delta), as a percentage, between the current window and its neighbors.

Related Topics

CMP Constraints

maxThicknessThreshold

Specifies the maximum copper thickness threshold, in angstroms, that determines which windows are considered thickness hotspots.

maxThicknessThreshold Quick Reference

Constraint Type	Layer
Value Types	FltValue
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxThicknessThreshold` constraints have a FltValue representing the maximum copper thickness threshold, in angstroms, that determines which windows are considered thickness hotspots. Any window with a thickness value greater than this maximum is considered to be a hotspot.

Related Topics

CMP Constraints

maxThicknessThresholdPercent

Specifies the maximum copper thickness threshold as a percentage of the measured thickness range. Any window whose thickness is less than this percentage below the maximum value of the measured thickness range is considered a hotspot.

maxThicknessThresholdPercent Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`maxThicknessThresholdPercent` constraints have a `FltValue` representing the maximum copper thickness threshold as a percentage of the measured thickness range.

Related Topics

CMP Constraints

minDensityGradient

Sets the minimum metal density gradient (delta), as a percentage, between the current window and its neighbors. This is used by CMP analysis.

minDensityGradient Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`minDensityGradient` constraints have a `FltValue` that represents the minimum metal density gradient as a percentage.

Related Topics

CMP Constraints

minThicknessGradient

Specifies the minimum copper thickness gradient (delta), as an absolute difference in angstroms, between the current CMP window and its neighbors.

minThicknessGradient Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`minThicknessGradient` constraints have a `FltValue` that specifies the minimum copper gradient (delta), as an absolute difference in angstroms, between the current CMP window and its neighbors.

Related Topics

CMP Constraints

minThicknessGradientPercent

Specifies the minimum copper thickness gradient (delta), as a percentage, between the current CMP window and its neighbors.

minThicknessGradientPercent Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`minThicknessGradientPercent` constraints have a `FltValue` that specifies the minimum copper gradient (delta), as a percentage, between the current CMP window and its neighbors.

Related Topics

CMP Constraints

minThicknessThreshold

Specifies the minimum thickness threshold, in angstroms, that determines which windows are considered thickness hotspots. Any window value below this minimum is considered to be a hotspot.

minThicknessThreshold Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`minThicknessThreshold` constraints have a `FltValue` that represents the minimum thickness threshold, in angstroms.

Examples

Format	Example
Tcl	`set_layer_constraint -constraint minThicknessThreshold \` `-layer` *`s_layer`* `-FltValue` *`f_thickness`*

Related Topics

CMP Constraints

minThicknessThresholdPercent

Specifies the minimum thickness threshold, as a percentage of the measured thickness range. Any window whose thickness is less than this percentage above the minimum value of the measured thickness range is considered to be a hotspot.

minThicknessThresholdPercent Quick Reference

Constraint Type	Layer
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	CMP

Value Type

`minThicknessThresholdPercent` constraints have a `FltValue` that represents the minimum thickness threshold, as a percentage of the measured thickness range.

Examples

Format	Example
Tcl	`set_layer_constraint -constraint minThicknessThresholdPercent \ -layer s_layer -FltValue f_thicknessPct`

Related Topics

CMP Constraints

7

Density Constraints

This topic lists the density constraints:

densityFillKeepout	maxDensity	maxDiffDensity
maxInterLayerDensityGradient	minDensity	minDensityHole
minFillPatternSpacing	minFillToFillSpacing	minPgFillBoundarySpacing
minPgFillClockSpacing	minPgFillFloatingFillSpacing	minPgFillPgSpacing
minPgFillSignalSpacing	minPgFillSpacing	

Related Topics

Tcl Constraint Commands

Manage Constraints

densityFillKeepout

Determines whether to prevent fill shapes from being added to fill keepout regions on a layer.

densityFillKeepout Quick Reference

Constraint Type	Layer
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, blockage, net group, group2group, reflexive, transreflexive, design, foundry
Category	Density

Value Type

`densityFillKeepout` constraints have a `BoolValue`. When set to `true`, fill shapes will not be added in fill keepout regions.

Related Topics

Density Constraints

maxDensity

Specifies the maximum percentage of the total design area the metal in the specified layer may occupy (`density = metal layer area/total design area`). The constraint can be given as a fixed value that applies to the entire design, or as a table of values that apply to areas of the design, as you step through them, and are based on the step size.

The minDensity constraint is used in conjunction with the `maxDensity` constraint to ensure that the metal density is neither too dense nor too sparse.

maxDensity Quick Reference

Constraint Type	Layer
Value Types	`FltValue`, `Flt1DTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Types

`maxDensity` constraints have the following value types:

- `FltValue`

  Specifies the maximum percentage of the design area that the metal on the specified layer may occupy over the entire area. The density percentage is calculated as the metal area divided by the total design area. Valid values: 0 to 100

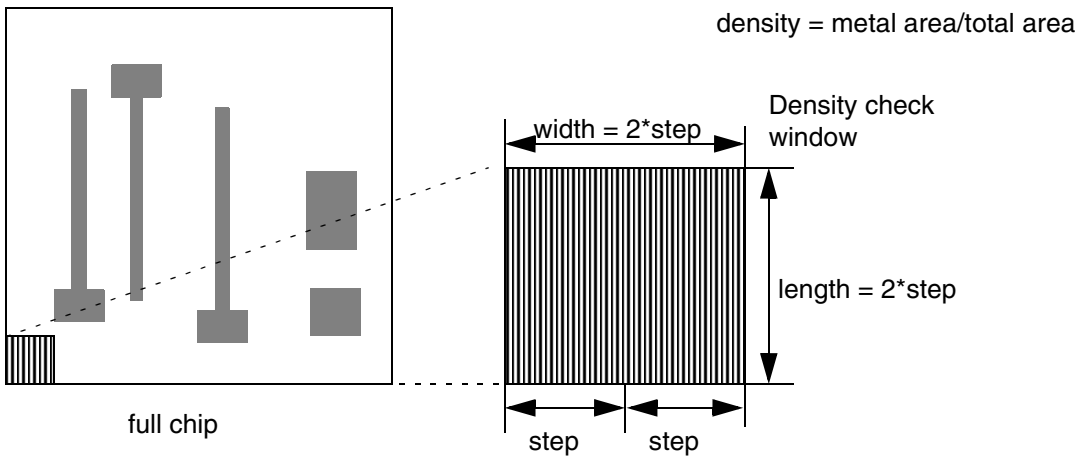
- `Flt1DTblValue`

  The lookup key (“step”) represents the step size and the value represents the maximum density percentage for a check window. Check windows are square with equal height and width.

  - If `windowStepSize` is not given, the check windows are 2*step size.
  - If `windowStepSize` is given, the window size that corresponds with the given step size is used.

  Density checking starts with the check window in a corner of the design, then the check window is moved across the design incrementally, by step size, each time validating that

the density percentage in the check window is less than or equal to the maximum allowed percentage.



Parameter

- `windowStepSize` (`OneDTblValue`, optional) The lookup key (“step”) represents the step size and the value represents the window size. This table is used only when `maxDensity` is a `Flt1DTblValue` to look up the window size associated with a given step size.

Examples

- Fixed Value

  Sets the maximum `Metal2` density to 70%.

Format	Example
Tcl	`set_layer_constraint -constraint maxDensity \` `-layer Metal2 -Value 70.0`
LEF	`LAYER METAL2` `MAXIMUMDENSITY 70.0 ;`

Virtuoso	`spacings (`
	`( maxDensity "Metal2" 70.0)`
	`)`

- 1D Table: Index step size

Sets the maximum `Metal2` density based on the step size of the check window.

Format	Example
Tcl	`set_layer_constraint -constraint maxDensity \` `-layer Metal2 -Flt1DTblValue {50 70.0 100 75.0}`

Virtuoso	`spacingTables(`
	`( maxDensity "Metal2"`
	`(( "step" nil nil ))`
	`(`
	`50.0 70.0`
	`100.0 75.0`
	`)`
	`)`
	`) ;spacingTables`

Related Topics

Density Constraints

maxDiffDensity

Specifies the maximum density difference between adjacent, non-overlapping density check windows.

maxDiffDensity Quick Reference

Constraint Type	Layer
Value Types	Flt1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`maxDiffDensity` constraints have a `Flt1DTblValue`. The lookup key (“step”) represents the step size and the value represents the maximum density difference percentage from 0 to 1, between adjacent, non-overlapping check windows. Check windows are square with equal height and width.

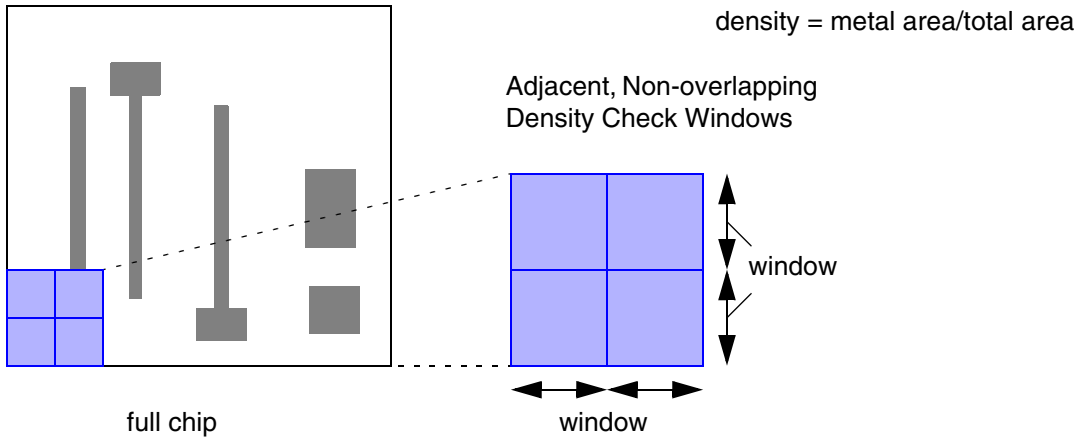
- If `windowStepSize` is not given, the check window dimensions are 2*step size.
- If `windowStepSize` is given, the window size that corresponds with the given step size is used.

Density checking starts with the check window in a corner of the design, then the check window is moved across the design incrementally, by step size, and the density is checked at each step.

Parameter

- `windowStepSize` (`OneDTblValue`, optional) The lookup key (“step”) represents the step size and the value represents the window size. This table is used to look up the

window size associated with a given step size. When this parameter is not given, window size is equal to 2*step size.



Examples

- 1D Table: Index step size

  Sets the maximum density difference percentage for `Metal2` based on the step size of the check window.

Format	Example

Tcl	`set_constraint_parameter -name windowStepSize -row_name window \`
	` -OneDTblValue { 60 120 120 240} -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down}`
	`set_layer_constraint -constraint maxDiffDensity -layer Metal2 \`
	` -Flt1DTblValue {60 0.1 120 0.2} -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down} -row_name step`

Virtuoso	`spacingTables(`
	`( maxDiffDensity "Metal2"`
	`(( "step" nil nil "window" nil nil))`
	`(`
	`( 60.0 120.0 ) 0.1`
	`(120.0 240.0 } 0.2`
	`)`
	`)`
	`) ;spacingTables`

Related Topics

Density Constraints

maxInterLayerDensityGradient

Specifies the maximum metal density gradient (delta), as a percentage, between the current layer and the layers above and below.

maxInterLayerDensityGradient Quick Reference

Constraint Type	Simple
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`maxInterLayerDensityGradient` constraints have a `FltValue` that represents the maximum metal density gradient (delta), as a percentage, between the current layer and the layers above and below.

Related Topics

Density Constraints

minDensity

Specifies the minimum metal density required for the layer, as a percentage from 0 to 100. The constraint can be given as a fixed value that applies to the entire design, or as a table of values that apply to areas of the design, as you step through them, and are based on the step size.

The maxDensity constraint is used in conjunction with the minDensity constraint to ensure that the metal density is neither too dense nor too sparse.

minDensity Quick Reference

Constraint Type	Layer
Value Types	`FltValue`, `Flt1DTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Types

`minDensity` constraints have the following value types:

- `FltValue`

  Specifies the minimum percentage of the design area that the metal on the specified layer may occupy over the entire area. The density percentage is calculated as the metal area divided by the total design area. Valid values: 0 to 100

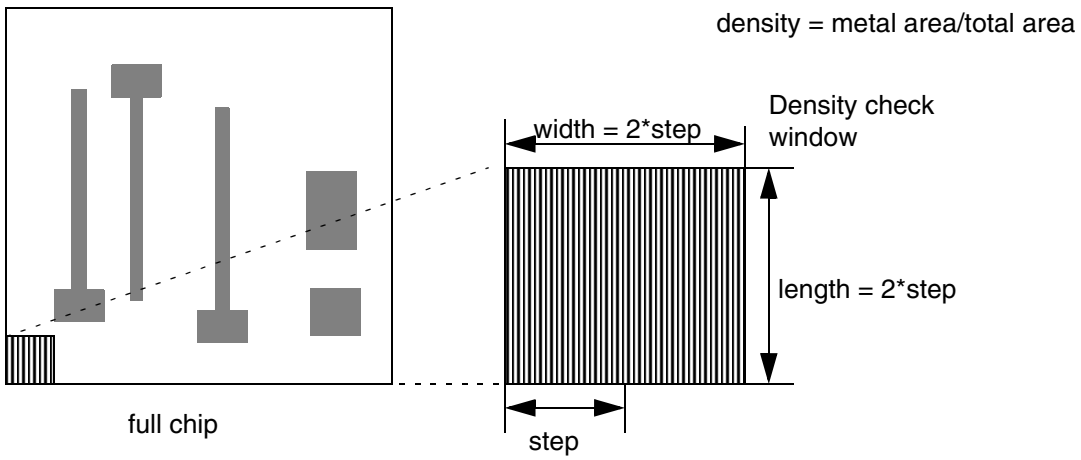
- `Flt1DTblValue`

  The lookup key (“step”) represents the step size and the value represents the minimum density percentage for a check window. Check windows are square with equal height and width.

  - If `windowStepSize` is not given, the check window dimensions are 2*step size.
  - If `windowStepSize` is given, the window size that corresponds with the given step size is used.

  Density checking starts with the check window in a corner of the design, then the check window is moved across the design incrementally, by step size, each time validating that

the density percentage in the check window is greater than or equal to the minimum allowed percentage.



Parameter

- `windowStepSize` (OneDTblValue, optional) The lookup key (“step”) represents the step size and the value represents the window size. This table is used only when `minDensity` is a `Flt1DTblValue` to look up the window size associated with a given step size.

Examples

- Fixed Value

  Sets the minimum `Metal2` density to 50%.

Format	Example
Tcl	`set_layer_constraint -constraint minDensity \` `-layer Metal2 -Value 50.0`
LEF	`LAYER Metal2` `MINIMUMDENSITY 50.0 ;`

Virtuoso	`spacings (`
	`( minDensity "Metal2" 0.50)`
	`)`

- 1D Table: Index step size

Sets the minimum `Metal2` density based on the step size of the check window.

Format	Example

Tcl	`set_constraint_parameter -name windowStepSize -row_name window \`
	` -OneDTblValue { 60 120 120 240} -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down}`
	`set_layer_constraint -constraint maxDiffDensity -layer Metal2 \`
	` -Flt1DTblValue {60 20.0 120 25.0} -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down} -row_name step`

Virtuoso	`spacingTables(`
	`  ( minDensity "Metal2"`
	`  (( "step" nil nil "window" nil nil))`
	`  (`
	`  ( 60.0 120.0 ) 20.0`
	`  (120.0 240.0 } 25.0`
	`  )`
	`  )`
	`) ;spacingTables`

Related Topics

Density Constraints

minDensityHole

Specifies the minimum effective width and minimum area for a hole on a layer. Holes that meet the given criteria can be identified using `check_density` and annotated for review. `create_fill` can be used to increase metal density in the holes by adding fill shapes.

minDensityHole Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minDensityHole` constraints have a `Value` that represents the minimum area for holes on the layer.

Parameter

- `width` (`Value`) specifies the minimum effective width for holes on the layer. This parameter is required.

Examples

This example identifies holes on `Metal2` that are at least 1.5 user units in width and at least 10 user units^2 in area.

```
set_constraint_parameter -name width -Value 1.5
set_layer_constraint -layer Metal2 -constraint minDensityHole -Value 10
```

Related Topics

Density Constraints

minFillPatternSpacing

Sets the minimum required distance between metal fill geometries and active geometries on a layer.

minFillPatternSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minFillPatternSpacing` constraints have a `Value` that represents the minimum required distance, in user units, between metal fill geometries and active geometries.

Examples

This example sets the minimum spacing between metal fill geometries and active geometries to 1.2 on `Metal1`.

```
set_layer_constraint -constraint minFillPatternSpacing -layer Metal1 -Value 1.2
```

Related Topics

Density Constraints

minFillToFillSpacing

Sets the minimum required distance, in user units, between metal fill geometries on a layer.

minFillToFillSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Density

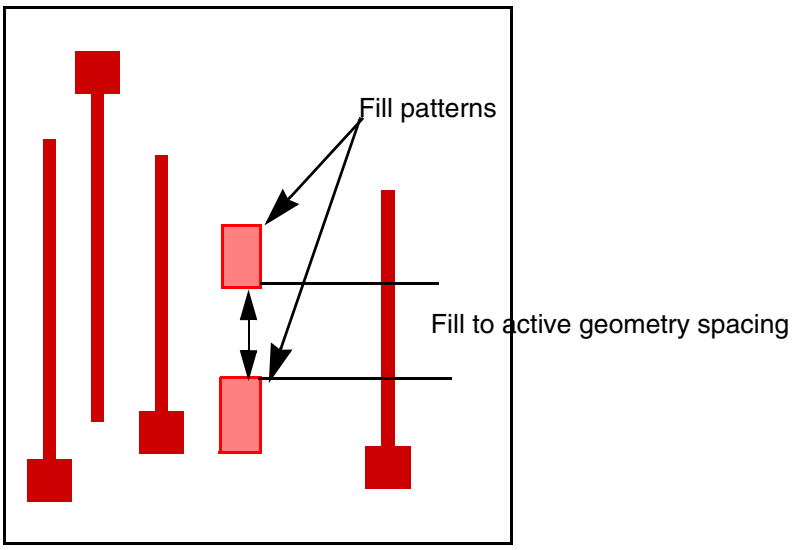
Value Type

`minFillToFillSpacing` constraints have a Value that represents the minimum required distance, in user units, between metal fill geometries.

Examples

This example sets the minimum spacing between metal fill geometries to 2.0 user units on `Metal1`.

Format	Example
Tcl	`set_layer_constraint -constraint minFillToFillSpacing \` `  -layer Metal1 -Value 2.0`
LEF	`PROPERTY LEF58_FILLTOFILLSPACING` `  "FILLTOFILLSPACING 2.0 ;" ;`



Related Topics

Density Constraints

minPgFillBoundarySpacing

Specifies the minimum spacing between power and ground fill stripes and the design boundary.

minPgFillBoundarySpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minPgFillBoundarySpacing` constraints have a `Value` that represents the minimum spacing, in user units, between power and ground fill stripes and the design boundary. If `minPgFillBoundarySpacing` is not specified, then minBoundaryInteriorHalo, if specified, is used, otherwise, the default spacing is 0.

Examples

This example sets the minimum spacing between power and ground fill stripes and the design boundary to 2.0 on `Metal1`.

```
set_layer_constraint -constraint minPgFillBoundarySpacing -layer Metal1 -Value 2.0
```

Related Topics

Density Constraints

minPgFillClockSpacing

Specifies the minimum spacing between power and ground fill stripes and clock nets.

minPgFillClockSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minPgFillClockSpacing` constraints have a `Value` that represents the minimum spacing, in user units, between power and ground fill stripes and clock nets. The default spacing is the minPgFillSignalSpacing constraint value.

Examples

This example sets the minimum spacing between power and ground fill stripes and clock nets to 2.0 on `Metal1`.

```
set_layer_constraint -constraint minPgFillClockSpacing -layer Metal1 -Value 2.0
```

Related Topics

Density Constraints

minPgFillFloatingFillSpacing

Specifies the minimum spacing, in user units, between power and ground fill stripes and existing floating fill shapes.

minPgFillFloatingFillSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minPgFillFloatingFillSpacing` constraints have a `Value` that represents the minimum spacing, in user units, between power and ground fill stripes and existing floating fill. The default spacing is the minPgFillSpacing value.

Examples

This example sets the minimum spacing between power and ground fill stripes and existing floating fill to 1.5 user units on `Metal1`.

```
set_layer_constraint -constraint minPgFillClockSpacing -layer Metal1 -Value 1.5
```

Related Topics

Density Constraints

minPgFillPgSpacing

Specifies the minimum spacing between power and ground fill stripes and existing power and ground nets.

minPgFillPgSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minPgFillPgSpacing` constraints have a `Value` that represents the minimum spacing, in user units, between power/ground fill stripes and existing power and ground nets. The default spacing is the minPgFillSignalSpacing constraint value.

Examples

This example sets the minimum spacing between power and ground fill stripes and existing power and ground nets to 2.0 on `Metal1`.

```
set_layer_constraint -constraint minPgFillPgSpacing -layer Metal1 -Value 2.0
```

Related Topics

Density Constraints

minPgFillSignalSpacing

Specifies the minimum spacing between power and ground fill stripes and signal nets.

minPgFillSignalSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minPgFillSignalSpacing` constraints have a `Value` that represents the minimum spacing, in user units, between power/ground fill stripes and existing power and ground nets. The default is minSpacing*2.

Examples

This example sets the minimum spacing between power and ground fill stripes and signal nets to 2.0 on `Metal1`.

```
set_layer_constraint -constraint minPgFillSignalSpacing -layer Metal1 -Value 2.0
```

Related Topics

Density Constraints

minPgFillSpacing

Specifies the minimum spacing between two power or ground fill stripes.

minPgFillSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Density

Value Type

`minPgFillSpacing` constraints have a `Value` that represents the minimum spacing, in user units, between two power or ground fill stripes. The default is the first defined value of the following: minSameNetSpacing*2, or minSpacing*2.

Examples

This example sets the minimum spacing between two power or ground fill stripes on `Metal1` to `0.6`.

```
set_layer_constraint -constraint minPgFillSpacing -layer Metal1 -Value 0.6
```

Related Topics

Density Constraints

8

Extension Constraints

This topic lists the extension constraints:

maxExtension	minBoundaryExtension	minCenterLineExtension
minConcaveCornerExtension	minDualEndOfLineExtension	minDualExtension
minEndOfLineExtension	minEndOfLineEdgeExtension	minExtension
minExtensionEdge	minExtensionOnLongSide	minExtensionToCorner
minInnerVertexProximityExtension	minNeighborExtension	minQuadrupleExtension
minSideExtension	minTouchingDirectionExtension	minViaJointExtension
minWireExtension	oaDummyPolyExtension	

Related Topics

Tcl Constraint Commands

Manage Constraints

maxExtension

Specifies the maximum distance a shape on one layer can extend past a shape on a second layer.

maxExtension Quick Reference

Constraint Type	Layer pair
Value Types	`Value`, `OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Types

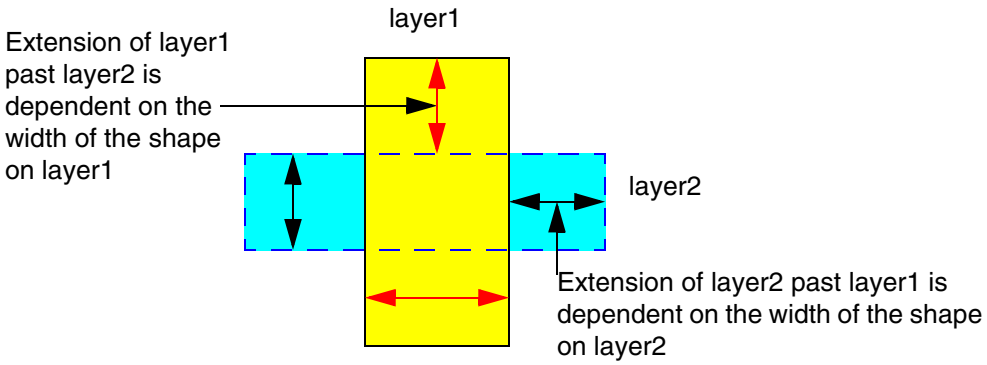
`maxExtension` constraints have the following value types:

- `Value`

  Specifies the maximum extension for a shape on one layer past a shape on another layer.

- `OneDTblValue`

  The lookup key for the table is the width of the shape on the second layer and the table value is the extension of the shape on the second layer past the shape on the first layer.



Examples

- Fixed Value

  Sets the maximum extension for `Metal2` past `Metal1` to 1.3.

Format	Example
Tcl	`set_layerpair_constraint -constraint maxExtension \` `-layer1 Metal1 -layer2 Metal2 -Value 1.3`

- 1D Table: Index width

  Sets maximum extension for `Metal2` past `Metal1`, based on the width of `Metal2`.

Format	Example

Tcl	`set_layerpair_constraint -constraint maxExtension \`
	`-layer1 Metal1 -layer2 Metal2 -row_name width \`
	`-OneDTblValue {0.6 0.4 0.8 0.7}`

Related Topics

Extension Constraints

minBoundaryExtension

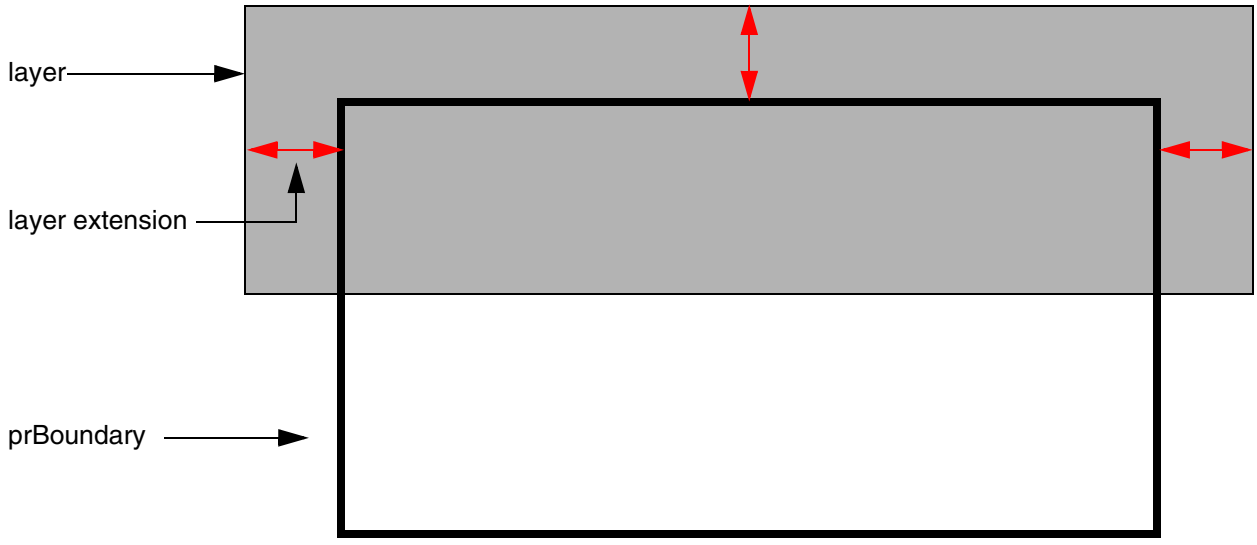
Specifies the minimum distance the specified layer can extend past the `prBoundary`. This constraint is commonly used to describe the necessary extension of a well past a boundary.

minBoundaryExtension Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry, boundary
Category	Extension

Value Type

`minBoundaryExtension` constraints have a `Value` that represents the minimum extension in user units that the shape must extend past the boundary.



The layer extension beyond the prBoundary must be greater than or equal to the value of this constraint.

Parameter

- `coincidentAllowed` (`BoolValue`, optional) specifies whether shapes must meet the minimum extension (`false`, default) or their edges can be coincident (`true`).

Examples

This example sets the minimum extension of `poly2` past the `prBoundary` to `1.5` user units.

Format	Example
Tcl	`set_layer_constraint -constraint minBoundaryExtension \` `-layer poly2 -Value 1.5`

Virtuoso	`spacings(`
	`(minPRBoundaryExtension "poly2" 1.5)`
	`)`

Related Topics

Extension Constraints

minCenterLineExtension

Specifies whether a centerline algorithm must be used to identify valid locations for via placement on a given cut layer and metal above or below. By definition, these locations guarantee that correct enclosures are available. The algorithm is applied to rectangles or polygons of connected/merged metal shapes.

The valid locations are based on via anchor points. By default, the center of a cut shape, regardless of dimension, is a valid anchor point.

The parameters for this constraint are used to specify the following:

- Alternate anchor points for rectangular cut classes and redundant cuts
- Joints and bends where vias are not allowed

minCenterLineExtension Quick Reference

Constraint Type	Layer pair
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

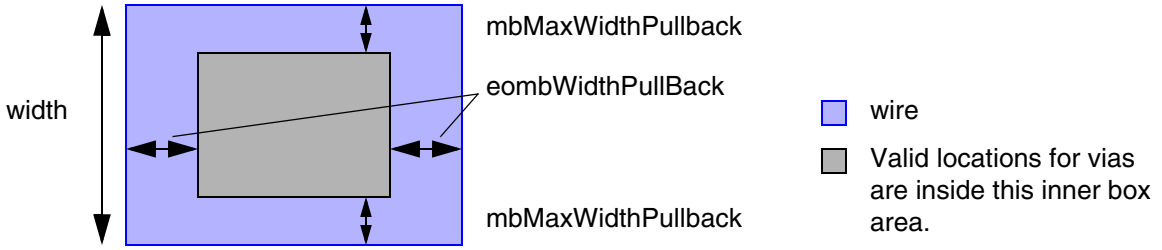
Value Type

`minCenterLineExtension` constraints have a `BoolValue`.

Parameters

- `centerLineDef` (`IntValue`, required) specifies which centerline algorithm to use. Currently, only a value of 1 is accepted and requires that the `mbMaxWidth` parameter also be set.
- `mbMaxWidth` (`Value`, required) specifies that the constraint applies for wire widths less than this value, in user units. This parameter is required when `centerLineDef` is set to 1.
- `cutClass` (`DualValue`, optional) specifies that the constraint applies for the given cut class width and height.

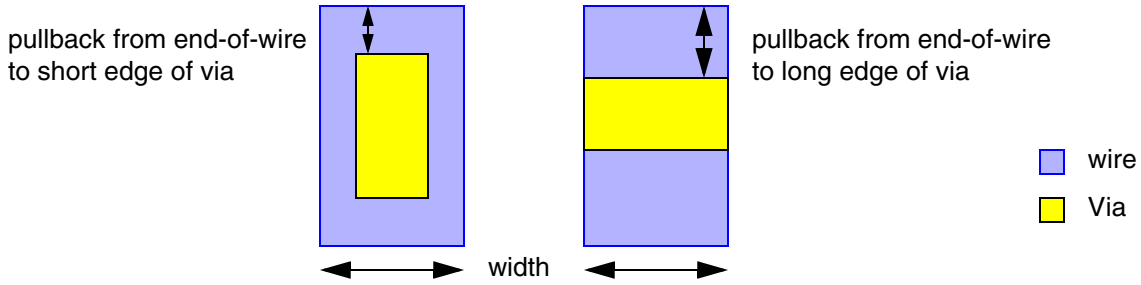
- `mbMaxWidthPullBack` (`DualValue`, optional) For wire widths greater than or equal to `mbMaxWidth`, the distance from a via edge to the side edge of the wire must be greater than or equal to the first pullback value, in user units.



  Note: Only the first value of the `DualValue` is used.

- `eombWidthPullBack` (`OneDDualValueTbl`, optional) specifies short edge and long edge pullbacks as a function of the wire's end-of-line width in the following format:

  `-OneDDualValueTbl {`*`f_width1 f_short1 f_long1 … f_widthn f_shortn f_longn`*`}`



  The distance from the end-of-wire edge to the via edge (short or long) must be greater than or equal to the given pullback distance.

- `eombWidthCoincidentOK` (`Int1DTblValue`, optional) specifies whether vias can be coincident with the end-of-line as a function of the wire's end-of-line width in the following format:

  `-Int1DTblValue {`*`f_width1 b_coincidentOK1 … f_widthn b_coincidentOKn`*`}`

- `numCuts` (`IntValue`, optional) specifies that the constraint applies only to a double-cut via. If more than one `minCenterLineExtension` constraints are defined for a cut class in an OR group, set `numCuts` to 2 to make those constraints mutually exclusive.

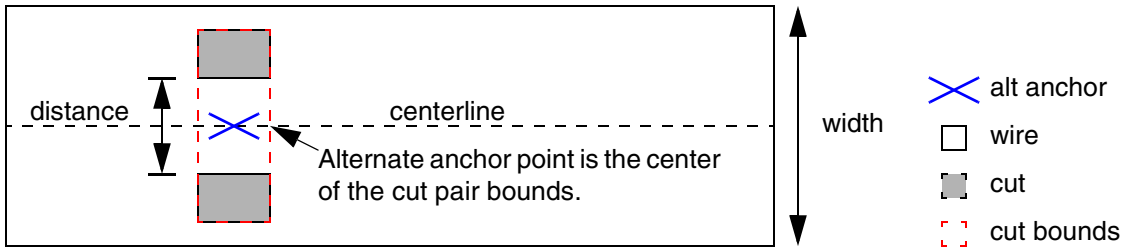
- `numCutDistance` (`Value`, optional) specifies that the constraint applies only if the distance between a via cut pair is less than or equal to this value. This parameter applies only when `numCuts` is set to `2`.

The following parameters specify conditions for alternate anchor points:

- `shortEdgeCenterAnchor` (`BoolValue`, optional). When set to `true`, the center of the short edge for a rectangular cut class is an alternate anchor point.
- `redundantCutCenterAnchor` (`ValueArrayValue`, optional) specifies a range of widths, in user units, and a distance, in user units, between two cuts in the following format:

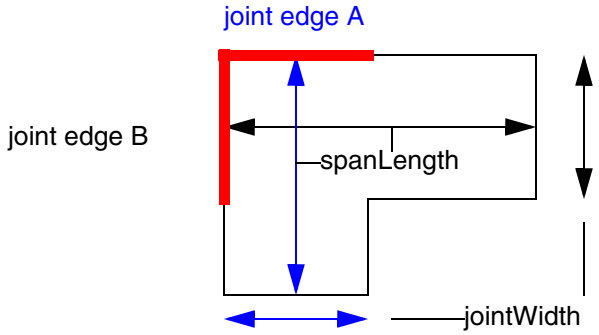
  `-ValueArrayValue {`*`f_width1 f_width2 f_distance`*`}`

  When two cuts are <= *`f_distance`* apart, and are aligned on the wire width >= *`f_width1`* and <= *`f_width2`*, an alternate anchor point is calculated as the center of the cut pair bounds, which must be on the wire's centerline. The alternate anchor point of the cut pair will be used if the cut anchor fails to fall on the centerline.



The following parameters determine the valid via locations at joints or bends:

- `jointWidth` (`Value`) specifies that the constraint applies only to joints whose width is less than this value, in user units. This parameter is required when `viaToJointDistance` or `viaToBothJointDistance` is given.
- `spanLength` (`Value`) specifies that the constraint applies only to joints whose joint length is greater than or equal to this value, in user units. This parameter is required when `viaToJointDistance` or `viaToBothJointDistance` is given.

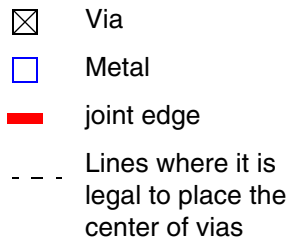


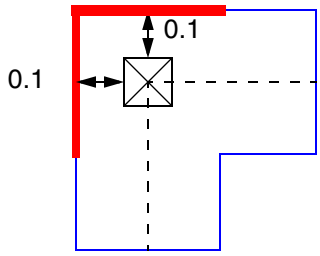
- `viaToJointDistance` (`Value`, optional) specifies the minimum spacing between the via and at least one joint edge, in user units.

- `viaToBothJointDistance` (`Value`, optional) specifies the minimum spacing between a via and both edges of a joint corner.

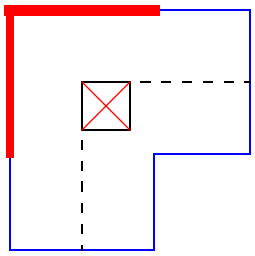
Typically, both `viaToJointDistance` and `viaToBothJointDistance` are specified. A violation occurs only when a via fails both conditions. `viaToJointDistance` is usually greater than `viaToBothJointDistance` and forces the via farther from the corner if one joint distance is small.

```
viaToBothJointDistance = 0.1
viaToJointDistance     = 0.2
```

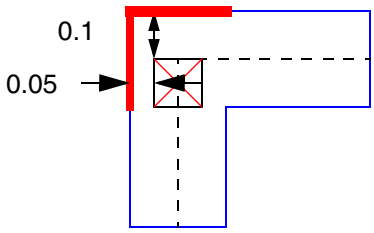




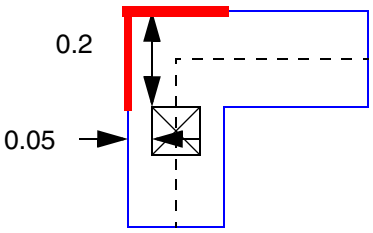
a) OK. Both edges of the via are >= `viaToBothJointDistance` from the joint edges and the center of the via is on centerline.



b) Violation. The center of the via is not on the metal centerline.



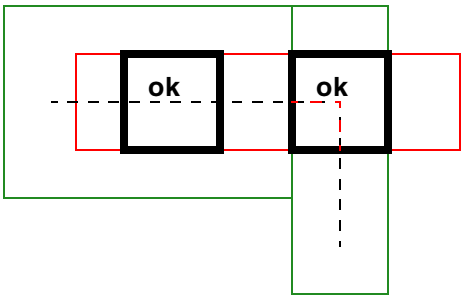
c) Violation. One via edge < `viaToBothJointDistance` from the joint edge and both via edges are < `viaToJointDistance` from the joint edges.



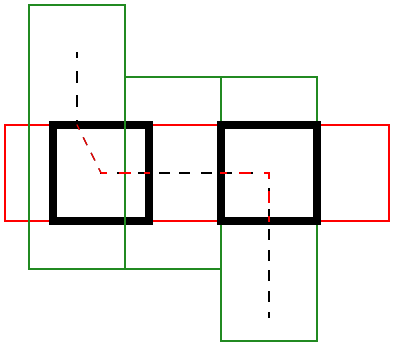
d) OK. One via edge < `viaToBothJointDistance` from the joint edge but the other via edge >= `viaToJointDistance` from a joint edge.

The following parameter determines when redundant cuts can cause an exception:

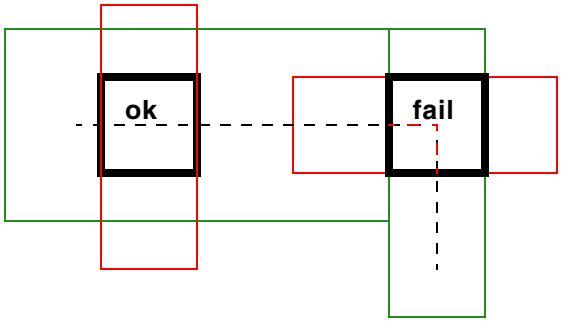
- `redundantCutDistance` (`Value`, optional) specifies a redundancy exception when there is an aligned redundant cut that is less than or equal to this distance, measured edge-to-edge in user units, from another cut on the same cut layer and on the same metal above and below with at least one common maximal rectangle above and below. No failure is reported if at least one of the redundant cuts meets the centerline requirements.



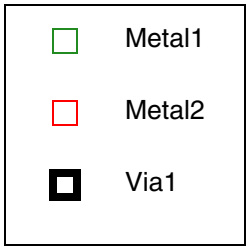
a) Allowed because left via meets the centerline requirements and the redundant right via is on the same metal above and below, within the required distance, and has a common shape on `Metal1` and `Metal2`.



b) Fails because neither via meets the centerline requirements.



c) Fails. The left via meets the centerline requirements but the right via fails and does not qualify for the redundancy exception because it does not have at least one common metal shape above and below.



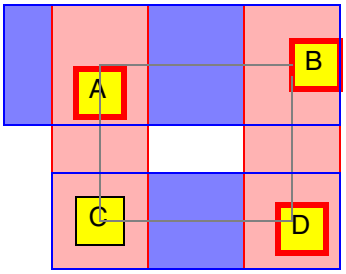
Examples

This example sets `minCenterLineExtension` constraints for three cut classes in an OR group for `Metal1/V2` and `Metal2/V2`. The parameter settings are the same for both metal/via pairs.

```
# For Metal1/V2

create_constraint_group -name m1_v2 -optype or -db design

# for cutClass 0.10 0.10 (VxLarge), see Figure minCenterLineExtension Example for
cutClass VxLarge.

set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.10 0.10}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue {0 0.025 0.025}
set_layerpair_constraint -constraint minCenterLineExtension \
  -layer1 Metal1 -layer2 V2 -symmetric false -BoolValue true -group m1_v2

# for cutClass 0.05 0.10 (VxBar), see Figure minCenterLineExtension Example for
cutClass VxLarge.

set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.05 0.10}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name redundantCutCenterAnchor \
  -ValueArrayValue {0.20 0.20 0.08}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue \
     {0    0.025 0.025 \
      0.10 0     0.025 \
      0.25 0.025 0.025}
set_layerpair_constraint -constraint minCenterLineExtension \
  -layer1 Metal1 -layer2 V2 -symmetric false -BoolValue true -group m1_v2

# for cutClass 0.05 0.05 (Vx), see Figure minCenterLineExtension Example for
cutClass VxLarge.

set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.05 0.05}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue \
     {0    0.05  0.05 \
      0.30 0.025 0.025}
set_constraint_parameter -name jointWidth -Value 0.15
set_constraint_parameter -name spanLength -Value 0.12
set_constraint_parameter -name viaToJointDistance -Value 0.05
set_constraint_parameter -name viaToBothJointDistance -Value 0.025
set_constraint_parameter -name redundantCutDistance -Value 0.15
set_layerpair_constraint -constraint minCenterLineExtension \
  -layer1 Metal1 -layer2 V2 -symmetric false -BoolValue true -group m1_v2

# For Metal2/V2

create_constraint_group -name m2_v2 -optype or -db design

# for cutClass 0.10 0.10 (VxLarge), see Figure minCenterLineExtension Example for
cutClass VxLarge.

set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.10 0.10}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue {0 0.025 0.025}
set_layerpair_constraint -constraint minCenterLineExtension \
  -layer1 Metal2 -layer2 V2 -symmetric false -BoolValue true -group m2_v2

# for cutClass 0.05 0.10 (VxBar), see Figure minCenterLineExtension Example for
cutClass VxLarge.
```

```
set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.05 0.10}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name redundantCutCenterAnchor \
  -ValueArrayValue {0.20 0.20 0.08}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue \
     {0    0.025 0.025 \
      0.10 0     0.025 \
      0.25 0.025 0.025}
set_layerpair_constraint -constraint minCenterLineExtension \
  -layer1 Metal2 -layer2 V2 -symmetric false -BoolValue true -group m2_v2

# for cutClass 0.05 0.05 (Vx), see Figure minCenterLineExtension Example for
cutClass VxLarge.

set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.05 0.05}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue \
     {0    0.05  0.05 \
      0.30 0.025 0.025}
set_constraint_parameter -name jointWidth -Value 0.15
set_constraint_parameter -name spanLength -Value 0.12
set_constraint_parameter -name viaToJointDistance -Value 0.05
set_constraint_parameter -name viaToBothJointDistance -Value 0.025
set_constraint_parameter -name redundantCutDistance -Value 0.15
set_layerpair_constraint -constraint minCenterLineExtension \
  -layer1 Metal2 -layer2 V2 -symmetric false -BoolValue true -group m2_v2
```

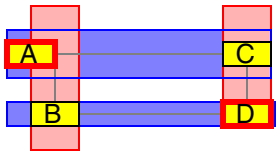
minCenterLineExtension Example for cutClass VxLarge



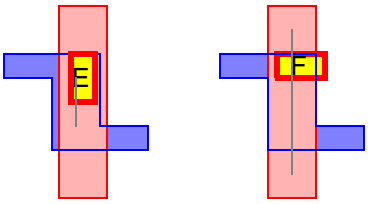
Via A: Violation. Center of via is not on `Metal1` centerline.
Via B: Violation. Center of via is not on `Metal2` centerline. The distance between the via edge and `Metal1` end-of-line is eombWidthPullBack.
Via C: OK. Via is on `Metal1` and `Metal2` centerlines and distance between via edge and `Metal2` end-of-line is >= eombWidthPullBack.
Via D: Violation. Center of via is not on `Metal1` or `Metal2` centerlines.


```
set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.10 0.10}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue {0 0.025 0.025}
```

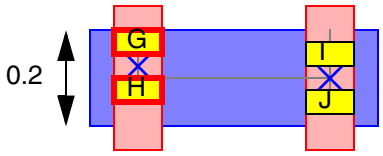
minCenterLineExtension Example for cutClass VxBar



Via A: Violation. Center of via is not on `Metal2` centerline.
Via B: OK. Cut class does not require `Metal1` extension for the short via edge on this `Metal1` width.
Via C: OK.
Via D: Violation. Cut class requires larger `Metal1` extension for end-of-line on this metal width and a larger extension for end-of-line for the long edge of the via on `Metal2`.



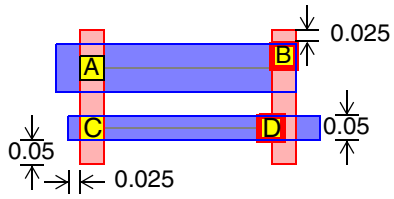
Via E: Violation. Center of via is not on `Metal1` centerline.
Via F: OK. Center of via is not on `Metal2` centerline.



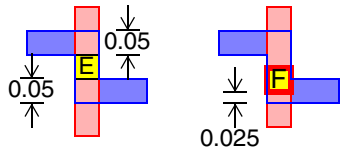
Vias G & H: Violation. Both vias are off the `Metal1` centerline. `Metal1` equals the redundantCutCenterAnchor width and the distance between vias <= redundantCutCenterAnchor distance. However, the center of the cut bounds is not on the `Metal1` centerline so both vias fail.
Vias I & J: OK. Both vias are off the `Metal1` centerline. However the alternate cut anchor passes because `Metal1` equals the redundantCutCenterAnchor width, the distance between vias <= redundantCutCenterAnchor distance and the center of the cut bounds is on the `Metal1` centerline.


```
set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.05 0.10}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name redundantCutCenterAnchor \
  -ValueArrayValue {0.20 0.20 0.08}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue \
     {0    0.025 0.025 \
      0.10 0     0.025 \
      0.25 0.025 0.025}
```

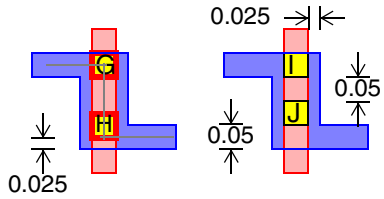
minCenterLineExtension Example for Vx



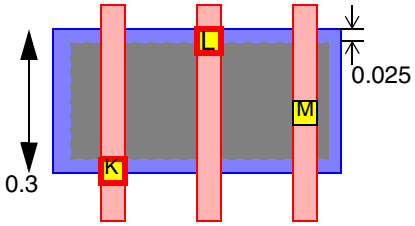
Via A: OK.
Via B: Violation. Center of via is not on `Metal1` centerline. `Metal2` extension < required 0.05.
Via C: Violation. `Metal1` extension < required 0.05.
Via D: Violation. Center of via is not on `Metal2` centerline.



Via E: OK. Via pullback from the top and bottom joints on `Metal1` is >= viaToJointDistance.
Via F: Violation. Via pullback from bottom `Metal1` joint is < viaToJointDistance.



Via G: Violation. No via pullback from top joint on `Metal1`.
Via H: Violation. Center of via is not on horizontal centerline of `Metal1`.
Via I & J: OK. Via I is in violation but via J is OK, and via I is <= redundantCutDistance from via J, so via I is allowed as a redundant



Via K: Violation. Via center is on edge of `Metal1`.
Via L: Violation. Via is aligned with wire edge and is partially outside the legal area.
Via M: OK. Via is > 0.025 from edge of `Metal1`.

legal locations for V2 via on `Metal1`

Metal1 | Metal2 | V2 OK | V2 Violation | Centerline

```
set_constraint_parameter -name centerLineDef -IntValue 1
set_constraint_parameter -name mbMaxWidth -Value 0.3
set_constraint_parameter -name cutClass -DualValue {0.05 0.05}
set_constraint_parameter -name mbMaxWidthPullBack -DualValue {0.025 0.025}
set_constraint_parameter -name eombWidthPullBack -OneDDualTblValue \
     {0    0.05  0.05 \
      0.30 0.025 0.025}
set_constraint_parameter -name jointWidth -Value 0.15
set_constraint_parameter -name spanLength -Value 0.12
set_constraint_parameter -name viaToJointDistance -Value 0.05
set_constraint_parameter -name viaToBothJointDistance -Value 0.025
set_constraint_parameter -name redundantCutDistance -Value 0.15
```

Related Topics

Extension Constraints

minConcaveCornerExtension

Sets the minimum extension of a shape on layer1 over a shape on layer2, as measured from the facing edge of layer2 to an inside corner on the layer1 shape. The minimum inside corner extension applies to an inside corner created by a geometry at any angle, diagonal or 90 degrees.

minConcaveCornerExtension Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`minConcaveCornerExtension` constraints have a `Value` that specifies the minimum extension of a shape on layer1 over a shape on layer2.

Examples

This example sets the minimum extension of a shape on `Metal1` over a shape on `Metal2`.

Format	Example
Tcl	`set_layerpair_constraint -constraint minConcaveCornerExtension \ -layer1 Metal1 -layer2 Metal2 -Value 1.0`
Virtuoso	`orderedSpacings( (minInsideCornerExtension "Metal1" "Metal2" 1.0) )`

Related Topics

Extension Constraints

minDualEndOfLineExtension

Some processes need to specify an end-of-line enclosure rule that applies only if no shapes on other nets lie in the rectangular regions around the side and end of the line. The two-layer `minDualEndOfLineExtension` specifies the extension of the first layer (layer1) beyond a shape on the second layer (layer2). The second layer is expected to be a cut layer.

This constraint is not symmetric.

minDualEndOfLineExtension Quick Reference

Constraint Type	Layer pair
Value Types	`DualValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Types

`minDualEndOfLineExtension` constraints have a `DualValue`. The first value is the extension, in user units, of layer1 beyond layer2 at the end-of-line (*eolExtension*). The same extension is also required on the opposite side of the layer2 shape. The second value is the extension, in user units, of layer1 beyond layer2 on either side (*otherExtension*).

Illustration of minDualEndOfLineExtension

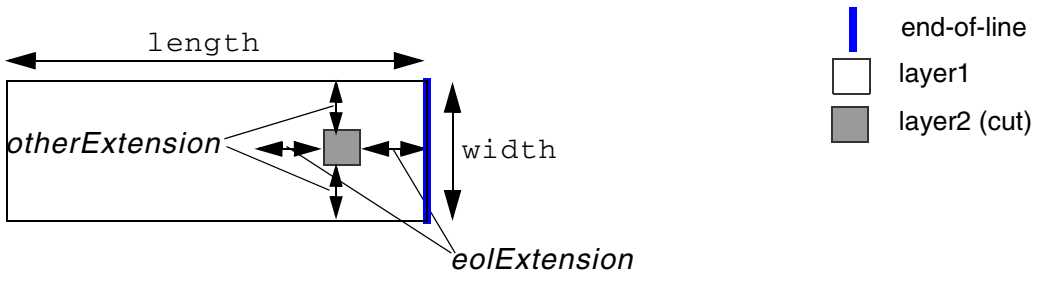
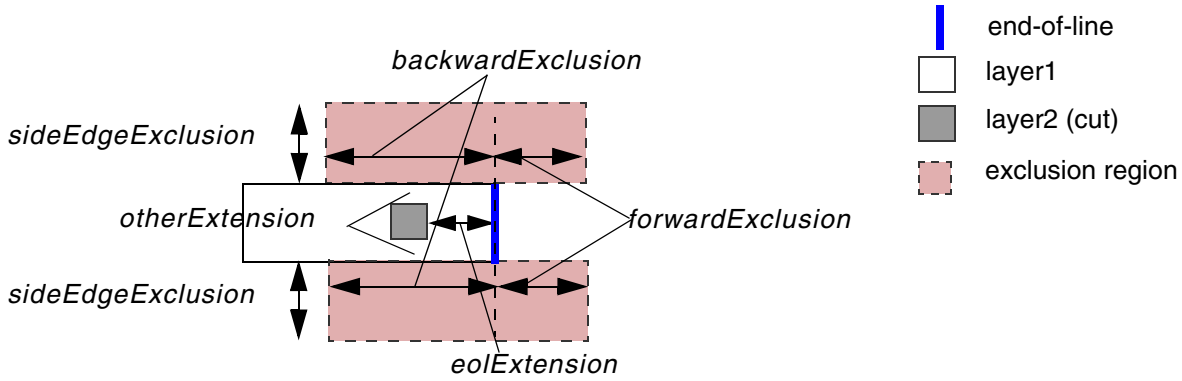
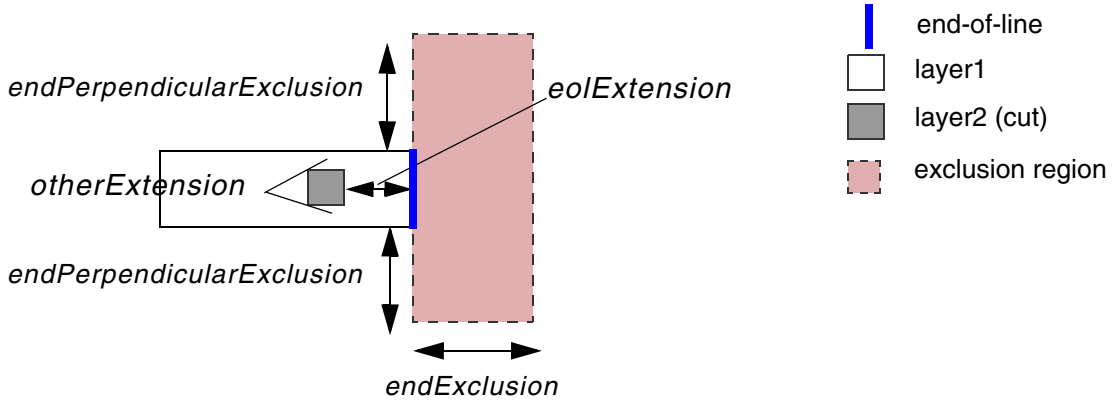


Illustration of minDualEndOfLineExtension sideExclusionArea



There should be no neighbor wires in the two exclusion regions in order to apply the given enclosure.

Illustration of minDualEndOfLineExtension endExclusionArea



There should be no neighbor wires in the exclusion region in order to apply the given enclosure.

Parameters

- `width` (`Value`, required) specifies that the constraint only applies if the width of the end-of-line is less than this value.
- `length` (`Value`, optional) specifies that the constraint only applies if the length of the end-of-line is greater than or equal to this value.
- `endOfLineOnly` (`BoolValue`, optional) If set `true` for a given constraint, it should be true for all minDualEndOfLineExtension constraints for the same cutClass and layer pair.

If only one constraint has the endOfLineOnly parameter, that constraint should be observed and the others ignored. If this parameter is `true`, then for all end-of-line edges, only `minDualEndOfLineExtension` constraints must be satisfied; any other enclosure type rules for that layer pair/cut class, such as `minDualExtension`, can be ignored.

- `cutClass` (`DualValue`, optional) identifies the cut class to which the constraint applies given the cut width and height.
- `endExclusionArea` (`DualValue`, optional) specifies that the constraint only applies if there are no different-net neighbors in the rectangular area at the end of the line. The first value is the distance on either side of the line end (*endPerpendicularExclusion*); the second value is the distance past the end-of-line (*endExclusion*) shown in Figure on page 345.
- `sideExclusionArea` (`ValueArrayValue`, optional) specifies an exclusion area at either side of the line. The area is defined by three Values—*sideEdgeExclusion*, *backwardExclusion*, and *forwardExclusion*—shown in Figure on page 345. The constraint only applies if there is no different-net neighbor shape in that region.
- `shortEdgeOnEol` (`BoolValue`, optional) When set to `true`, specifies that the first value (*eolExtension*) must be applied to the end/short edges of the cut and the second value (*otherExtension*) must be applied to the long sides of the cut. This parameter must be used with the `cutClass` parameter for rectangular cut vias.

Examples

Example 1—minDualEndOfLineExtension with endExclusionArea and sideExclusionArea

In this example, an `Metal1` or `Metal2` extension of 0.1 μm on all sides of a V1 cut is required. However, if the cut is on an end-of-line, then an extension of 0.15 μm is required to the end-of-line edge and on the opposite side of the cut, and 0.05 μm is required on the sides of the cut if there are no neighbor wires within 0.04 μm of the line side edges with back and forward

exclusion extensions of 0.03 μm or within 0.08 μm of the end-of-line edge with a perpendicular extension of 0.06 μm to the sides.

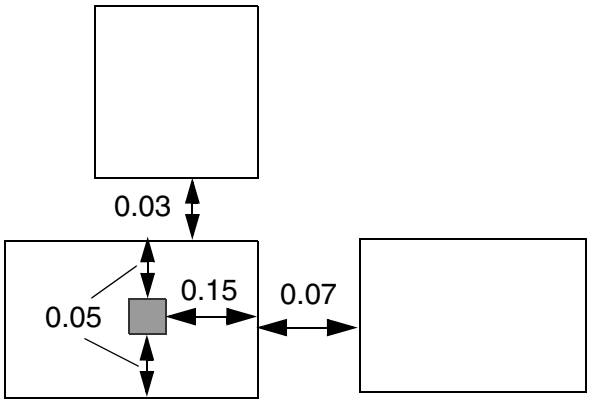
Format	Example

Tcl	`set_layerpair_constraint -constraint minDualExtension -layer1 Metal1 -layer2 V1 -DualValue {0.1 0.1}`
	`set_layerpair_constraint -constraint minDualExtension -layer1 Metal2 -layer2 V1 -DualValue {0.1 0.1}`
	`set_constraint_parameter -name width -Value 0.2`
	`set_constraint_parameter -name endExclusionArea -DualValue {0.08 0.06}`
	`set_constraint_parameter -name sideExclusionArea -ValueArrayValue {0.04 0.03 0.03}`
	`set_layerpair_constraint -constraint minDualEndOfLineExtension \`
	`  -layer1 Metal1 -layer2 V1 -DualValue {0.15 0.05}`
	`set_constraint_parameter -name width -Value 0.2`
	`set_constraint_parameter -name endExclusionArea -DualValue {0.08 0.06}`
	`set_constraint_parameter -name sideExclusionArea -ValueArrayValue {0.04 0.03 0.03}`
	`set_layerpair_constraint -constraint minDualEndOfLineExtension \`
	`  -layer1 Metal2 -layer2 V1 -DualValue {0.15 0.05}`

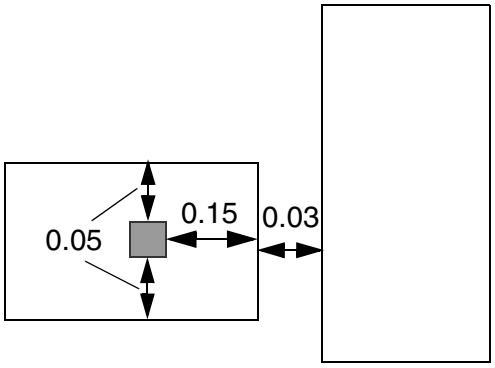
LEF	`ENCLOSURE 0.1 0.1`
	`PROPERTY LEF58_ENCLOSURE`
	`"ENCLOSURE EOL 0.2 0.15 0.05 SIDESPACING 0.04 EXTENSION 0.03 0.03 ;`
	` ENCLOSURE EOL 0.2 0.15 0.05 ENDSPACING 0.08 EXTENSION 0.06 ; " ;`

Virtuoso	`orderedSpacings(`
	`( minOppExtension "Metal1" "V1"  (0.1 0.1))`
	`( minOppExtension "Metal2" "V1"  (0.1 0.1))`
	`( minOppEndOfLineExtension "Metal1" "V1"`
	`  'width 0.2`
	`  'endExclusion  (0.08 0.06)`
	`  'sideExclusion (0.04 0.03 0.03)`
	`  (0.15 0.05))`
	`( minOppEndOfLineExtension "Metal2" "V1"`
	`  'width 0.2`
	`  'endExclusion  (0.08 0.06)`
	`  'sideExclusion (0.04 0.03 0.03)`
	`  (0.15 0.05))`
	`) ;orderedSpacings`

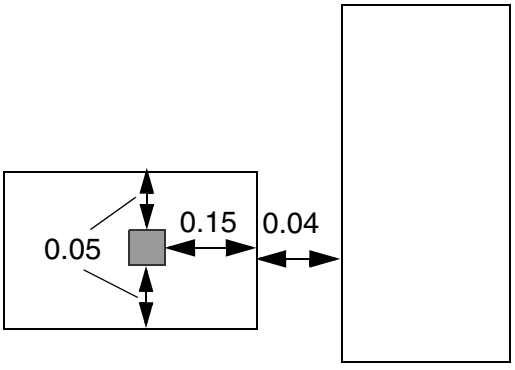
Illustrations for minDualEndOfLineExtension with endExclusionArea and sideExclusionArea Example



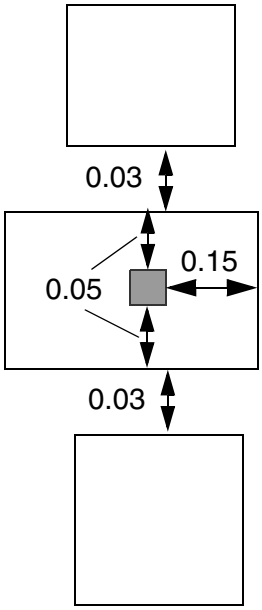
a) Violation, There are neighbor wires of orthogonal edges in both the end and side exclusion regions of the end-of-line, so the `minDualEndOfLineExtension` does not apply. An enclosure of 0.1 on all sides of the cut is required and not met (0.05< 0.1).



b) Violation. The neighbor wire opposite the end-of-line edge is within the 0.08 end exclusion region, and within the forward extension of the side region, so enclosure of 0.1 is required on all sides of the cut but not met.



c) OK, the neighbor wire is greater than 0.04 from the top/bottom edges. 0.15 and 0.05 enclosure should be applied, and is met.



d) OK, as long as the wires are not neighbors of orthogonal edges, the 0.15 and 0.05 enclosure can be applied, and is met.

Example 2—minDualEndOfLineExtension with shortEdgeOnEol

In this example, the end/short edge of a VB cut (0.04 x 0.08 rectangular cut) on cut layer V1 must have a `Metal1` or `Metal2` extension of 0.05 user units and the long sides must have an extension of 0.04 user units.

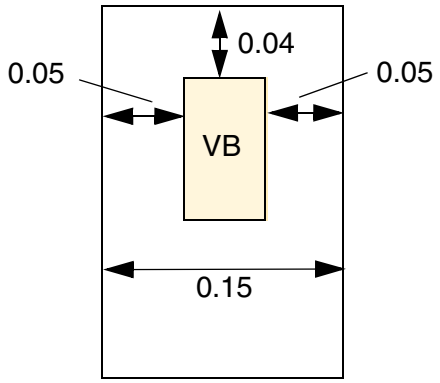
Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.2`
	`set_constraint_parameter -name cutClass -DualValue {0.04 0.08}`
	`set_constraint_parameter -name shortEdgeOnEol -BoolValue true`
	`set_layerpair_constraint -constraint minDualEndOfLineExtension \`
	`  -layer1 Metal1 -layer2 V1 -DualValue {0.05 0.04}`
	
	`set_constraint_parameter -name width -Value 0.2`
	`set_constraint_parameter -name cutClass -DualValue {0.04 0.08}`
	`set_constraint_parameter -name shortEdgeOnEol -BoolValue true`
	`set_layerpair_constraint -constraint minDualEndOfLineExtension \`
	`  -layer1 Metal2 -layer2 V1 -DualValue {0.05 0.04}`

LEF	`PROPERTY LEF58_ENCLOSURE`
	`"ENCLOSURE CUTCLASS VB EOL 0.2`
	`  SHORTEDGEONEOL 0.05 0.04 ;" ;`

Virtuoso	`orderedSpacings(`
	`( minOppEndOfLineExtension Metal1 V1`
	`  'width 0.2`
	`  'cutClass 0.04 0.08`
	`  'shortEdgeOnEol`
	`  0.05 0.04)`
	`( minOppEndOfLineExtension Metal2 V1`
	`  'width 0.2`
	`  'cutClass 0.04 0.08`
	`  'shortEdgeOnEol`
	`  0.05 0.04)`
	`) ;orderedSpacings`

Illustration of minDualEndOfLineExtension with shortEdgeOnEol Example



a) Violation. The end/short edge extension at the end-of-line edge is 0.04 < the required 0.05 user units.

Related Topics

Extension Constraints

check_extension

minDualExtension

Specifies, in user units, the minimum distance a shape on a routing layer must extend past a rectangle on a cut layer. The constraint specifies a pair of values which represents the required overhang in the horizontal and vertical directions and can be given in any order.

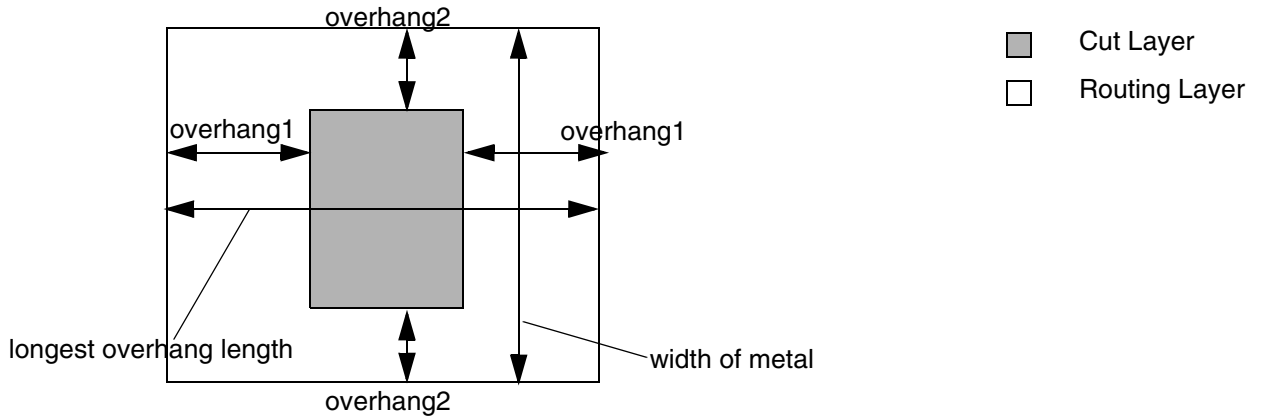
This constraint can specify a single pair of values (`DualValue`), or multiple pairs of values (`OneDDualArrayTblValue` or `Dual1DTblValue`) that are keyed by the width of the metal wire.

Optional Parameters can be used to qualify when the constraint applies. This constraint is not symmetric.

If you specify a `minDualExtension` for a layer pair, you should not specify a `minExtension` constraint.

In some processes, enclosure rules may only apply for rectangular cuts that do not collide with the centerline of the wire (`offCenterline`). In addition, in end-of-line enclosure rules which specify two enclosure values, it may be necessary for rectangular cut classes to specify that the end-of-line enclosure value is applied to the short edges, and the other overhang value should be applied to the long edges (`endSideOverhang`).

Enclosure Rule



Usually, an OR constraint group should contain only settings of the same constraint with different parameters. If any of the settings is satisfied, the constraint requirement is met. The exception is that minQuadrupleExtension, minDualExtension, and minWireExtension can be bound together in an OR group and if any of the constraints is satisfied, the extension requirement is met.

minDualExtension Quick Reference

Constraint Type	Layer pair
Value Types	DualValue, Dual1DTblValue, OneDDualArrayTblValue
Database Types	Design, Technology
Scope	net (for create_derived_vias only), design, foundry
Category	Extension
Group Operators	AND, OR

Value Types

`minDualExtension` constraints have the following value types:

- DualValue represents the minimum extension in each direction in user units (for example, `{0.2 0.5}` requires an extension of 0.2 in one direction and 0.5 in the opposite direction).
- Dual1DTblValue is two 1-D tables of values.

  The first table has width/extension pairs. The lookup key (`width`) represents the width of a shape on the first layer and the value represents the minimum extension of that first shape past a shape on the second layer. For example, `{widthA extA1 widthB extB1…}`.

  The second table is extension-based extensions. The lookup key (`extension`) is the minimum extension of a layer1 shape past a layer2 shape in one direction (horizontal or vertical) taken from the extension value in the first table, and the value is the minimum extension in the opposite direction. For example, {`extA1 extA2 extB1 extB2…`}, where the value of `extA1` is the same in both of the 1-D tables.
- OneDDualArrayTblValue uses width values as the lookup key for the table of arrays. For each width, there is a count for the number of extension pairs associated with the width, and the extension pairs.

  {*f_width1 i_numPair1* {*f_ext f_ext*}… {*f_ext f_ext*}$_{i\_numPair1}$
  f_width2 i_numPair2 {*f_ext f_ext*}… {*f_ext f_ext*}$_{i\_numPair2}$ …
  f_widthN i_numPairN {*f_ext f_ext*}… {*f_ext f_ext*}$_{i\_numPairN}$**}**

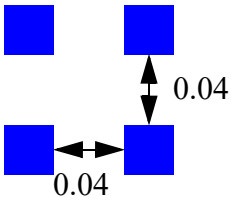
  If you specify multiple enclosure rules with different `minWidth` values, the largest `minWidth` rule that is still less than or equal to the wire width applies. For example, if you specify enclosure rules for 0.0 μm and 2.0 μm widths, then a 0.5 μm wire must meet a 0.0 μm rule, and a 2.0 μm wire must meet a 2.0 μm rule. A width of 0 implies that the extensions apply to minimum width wires. Widths greater than 0 imply extensions for wide wires of the given width.

Parameters

The `minDualExtension` constraint has the following parameters:

- `coincidentAllowed` (`BoolValue`, optional) specifies whether shapes must meet the minimum extension or their edges can be coincident (`true`). By default and when set to `false`, shapes must meet the minimum extension requirements and the edges cannot be coincident.
- `cutClass` (`DualValue`, optional) specifies that the constraint only applies to cut shapes whose width and length are equal to the values in the `DualValue` pair, specified in that order. Cut class applies to 32nm rules.
  - `offCenterline` (`BoolValue`, optional) When `true`, the constraint only applies to rectangular cuts that do not collide with the centerline of the wire. Must be used with the `cutClass` parameter.
  - `endSideOverhang` (`BoolValue`, optional) When `true`, the first extension value corresponds to the overhang of the end (short edge), and the second value corresponds to the overhang of the side (long edge). This parameter only applies to rectangular cut class constraints.
- `sizedBy` (`Value`, optional/custom) sizes (shrinks) extension values, in user units.
- `length` (`Value`, optional) specifies that the constraint only applies if the longest overhang length is greater than or equal to this parameter value.
- `numCuts` (`IntValue`, optional) specifies the minimum required number of same metal cuts for which this constraint applies (default: 1).
- `redundantCutDistance` (`Value`, optional) specifies that the constraint applies only if there is a redundant cut that is less than or equal to this distance from the original cut with the same metal shape above and below, and the redundant cut has the required enclosures for a cut given by a `minDualExtension` constraint without a specified `redundantCutDistance` parameter. The `minDualExtension` constraint value with the `redundantCutDistance` parameter will typically have smaller enclosure requirements.
- `atLeastOne` (`BoolValue`, optional) indicates that if at least one cut passes the `minDualExtension` constraint and there exists another `minDualExtension` constraint with `redundantCutDistance` defined, then any other cut within that distance satisfying the constraint with `redundantCutDistance` passes the via extension check.
- `except2x2Cuts` (`Value`, optional) specifies that an aligned 2 x 2 array of cuts whose vertical and horizontal distance is less than or equal to this distance will apply and pass

the constraint without any further checking. It is best not to put this parameter on a constraint with the `redundantCutDistance` parameter.




```
set_constraint_parameter -name except2x2Cuts -value 0.04
```

- `exceptCutMetalEdgeExtension` (`BoolValue`, optional) When `true`, if a metal shape of a cut via protrudes on a certain side of the wide wire, then the enclosure is not checked on that side. In addition, the enclosure based on the width of the protrusion must also be checked against the cut.
- `oaSpacingDirection` (`IntValue`, optional) is an integer specifying the direction in which the first extension value applies; the second value applies in the opposite direction. Valid values:
  0 any
  1 horizontal
  2 vertical
- `stepSizePair` (`DualValue`, optional) increases the required extension using the formula below:

  ```
  constraint value + n*stepSizePair
  ```

  The order of the two numbers corresponds to the order in the constraint value. If an entry is zero, then no stepped extension applies in that direction.
- Width-based constraint values of `OneDDualArrayTblValue` or `Dual1DTblValue` specify the extension requirements for wires that are greater than or equal to a given width value. A width of 0 implies that the extensions apply to minimum width wires. Widths greater than 0 imply extensions for wide wires of the given width. Some processes define *exceptions* to the via enclosure constraints provided that cut shapes on the wire shapes are sufficiently close, known as redundant vias. If you specify multiple enclosure rules with different *`minWidth`* values, the largest *`minWidth`* rule that is still less than or equal to the wire width applies. The following parameters can be used with wide wire `minDualExtension` values to specify conditions for redundant via exceptions:
  - `oaCutDistance` (`Value`, optional) specifies that if there is a neighbor cut that is less than or equal to this distance from a cut, then the neighbor cut is considered to be a redundant via.

The following parameters are **mutually exclusive**:

- `oaNoSharedEdge` (`BoolValue`, optional) determines whether vias on a shared edge that are within `oaCutDistance` can be counted as redundant. If the value of this parameter is `true`, only cuts that are less than or equal to `oaCutDistance` away but *not* along the same wire edge are considered to be redundant. The default of this parameter is `false`, meaning that cuts along the same wire edge and within `oaCutDistance` can be counted as redundant.
- `hasParallelRunLength` (`BoolValue`, optional) specifies that the constraint does not apply if there is a redundant cut less than or equal to `oaCutDistance` away with a parallel run length > 0 on the opposite edges for all failed edges of the original cut shape which do not satisfy the minimum extension requirements. The default for this parameter is `false`.

- `parallelRunLengthWidth` (`BoolValue`, optional), if set to `true`, both cuts must be covered by a maximal rectangle that maps to the same width bucket. The default for this parameter is `false`.

Redundant Cut Exceptions for minDualExtension

Redundant cut exists?	oaNoSharedEdge	hasParallelRunLength	numCuts=1	numCuts=2
yes, <= oaCutDistance from original cut	`false`	`false`	Constraint passes and no checking is done	Constraint applies and is checked for the original cut
no	`false`	`false`	Constraint applies and is checked for the original cut	Constraint does not apply
yes, <= oaCutDistance from original cut with no shared edge	`true`	`false`	Constraint passes and no checking is done	Constraint applies and is checked for the original cut
no, there is no cut <= oaCutDistance from original cut with no shared edge	`true`	`false`	Constraint applies and is checked for the original cut	Constraint does not apply

Redundant Cut Exceptions for minDualExtension

Redundant cut exists?	oaNoSharedEdge	hasParallelRunLength	numCuts=1	numCuts=2
yes, <= oaCutDistance from original cut with PRL > 0 on opposite edges for all edges of the original cut that do not satisfy zero width constraint value	`false`	`true`	Constraint passes and no checking is done	Constraint applies and is checked for the original cut against the effective width of the wire that it is touching
no, there is no cut <= oaCutDistance from original cut with PRL > 0 on opposite edges for all edges of the original cut that do not satisfy zero width constraint value	`false`	`true`	Constraint applies and is checked for the original cut	Constraint does not apply

Examples

Example 1—minDualExtension with DualValue

This example sets ground rules for `Metal1` shapes overlapping via1 and `Metal2` shapes overlapping via1.

Format	Example

Tcl	`set_layerpair_constraint -layer1 Metal1 -layer2 Via1 \`
	`  -constraint minDualExtension -hardness hard -DualValue {0.05 0.20}`
	`set_layerpair_constraint -layer1 Metal2 -layer2 Via1 \`
	`  -constraint minDualExtension -hardness hard -DualValue {0.10 0.10}`

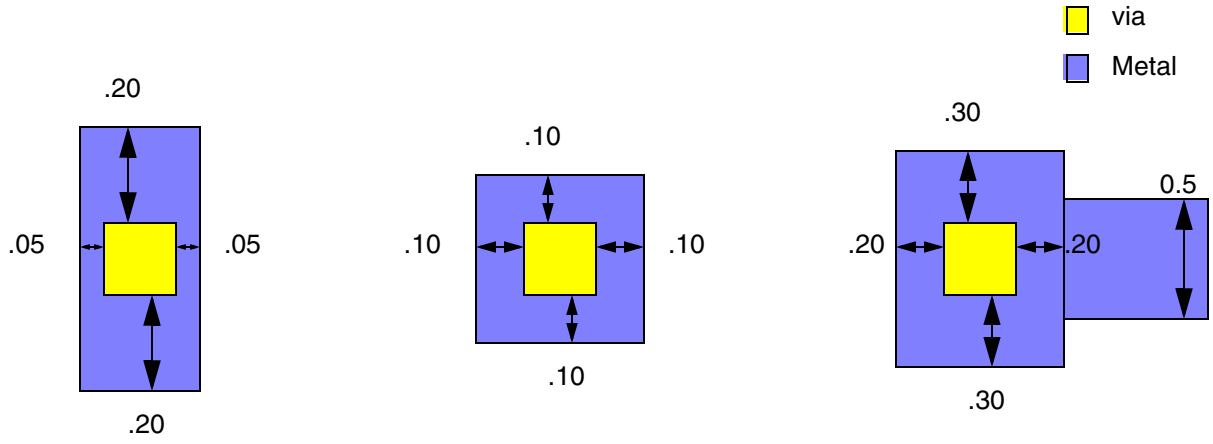
Format	Example

LEF	`LAYER Via1`
	`  ENCLOSURE ABOVE 0.10 0.10 ;`
	`  ENCLOSURE BELOW 0.05 0.20 ;`

Virtuoso	`orderedSpacings (`
	`  ( minOppExtension "Metal2" "Via1" ( 0.10 0.10 ))`
	`  ( minOppExtension "Metal1" "Via1" ( 0.05 0.20 ))`
	`)`

Example 2—minDualExtension with OneDDualArrayTblValue

This example sets three ground rules for `Metal1` shapes overlapping via1. For a `Metal1` width of 0, two extension pairs are given, {0.05 0.20} and {0.10 0.10}. For `Metal1` widths greater than or equal to 0.5, extensions must be 0.20 and 0.30 on opposite sides. The OneDDualArrayTblValue for this example is represented in Table on page 358.



Ground Rule 1
Metal1 must overlap via1 on two opposite sides by a minimum of 0.05 and must overlap the other two sides by 0.20

Ground Rule 2
Metal1 must overlap via1 on two opposite sides by a minimum of 0.10 and overlap the other two sides by 0.1

Ground Rule 3
Metal1 with a width>=0.5 must overlap via1 on two opposite sides by a minimum of 0.2 and must overlap the other two sides by 0.30

To set the `minDualExtension` constraint for this example, you would use the following:

Format	Example

Tcl	`set_layerpair_constraint -layer1 Metal1 -layer2 via1 \`
	`  -constraint minDualExtension -hardness hard -row_name width \`
	`  -OneDDualArrayTblValue {0 2 0.05 0.20 0.10 0.10 0.5 1 0.20 0.30}`

Format	Example

LEF	`LAYER via1`
	`ENCLOSURE BELOW 0.05 0.20 ;`
	`ENCLOSURE BELOW 0.10 0.10 ;`
	`ENCLOSURE BELOW 0.20 0.30 WIDTH 0.5 ;`

Example of minDualExtension OneDDualArrayTblValue

Width	Number of Extension Pairs	Extension Pairs
0.0	2	{ 0.05 0.20 } { 0.10 0.10 }
0.5	1	{ 0.20 0.30 }

Example 3—minDualExtension with cutDistance

In this example, if the wire width is greater than or equal to 0.3 μm, then an enclosure of 0.03 μm in all directions is required except when there is a redundant cut within 0.2 μm, then a {0.0 0.05} or {0.01 0.04} enclosure is needed. Cuts along the same edge and within

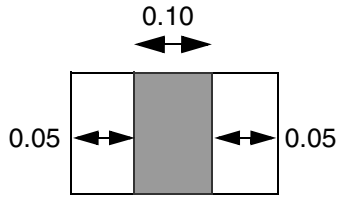
`cutDistance` can be considered to be redundant (by default, the `oaNoSharedEdge` parameter is false).

Format	Example

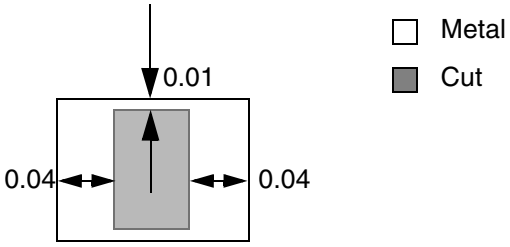
Tcl	`set_layer_constraint -constraint minWidth -layer via34 -Value 0.10`
	`set_layer_constraint -constraint minSpacing -layer via34 -Value 0.10`
	
	`create_constraint_group -name Ext34 -opType or`
	
	`#minDualExtension settings for m3 layer below via34`
	
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0  2 0.0  0.05  0.01 0.04 }`
	
	`set_constraint_parameter -name numCuts -IntValue 2`
	`set_constraint_parameter -name oaCutDistance -Value 0.2`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0.3 1 0.03 0.03}`
	
	`#minDualExtension settings for m4 layer above via34`
	
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0  2 0.0  0.05  0.01 0.04 }`
	
	`set_constraint_parameter -name numCuts -IntValue 2`
	`set_constraint_parameter -name oaCutDistance -Value 0.2`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0.3 1 0.03 0.03}`

LEF	`LAYER via34`
	`TYPE CUT ;`
	`WIDTH  0.10 ;  #minimum size of a cut`
	`SPACING 0.10 ;  #minimum edge-to-edge spacing is 0.10`
	`ENCLOSURE 0.0  0.05 ;  #overhang  0.0  0.05`
	`ENCLOSURE 0.01 0.04 ;  #or, overhang 0.01 0.04`
	`#if metal width >= 0.3, need 0.03 0.03, unless extra cut across`
	`# wire within 0.2µm; applies to m4 and m3 layers above and below`
	`ENCLOSURE 0.03 0.03 WIDTH 0.3 EXCEPTEXTRACUT 0.2 ;`
	`...`
	`END via34`

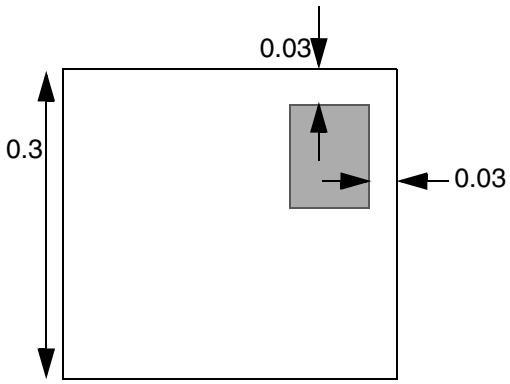
Illustration for minDualExtension with cutDistance Example



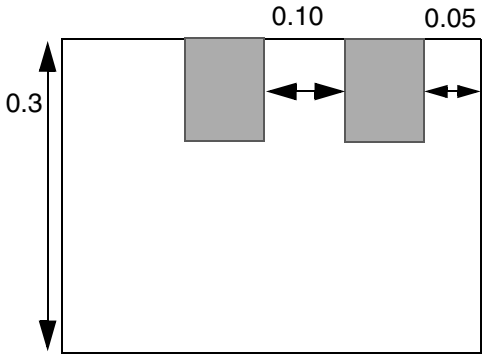
a) OK; has 0.0 and 0.05 overhang.



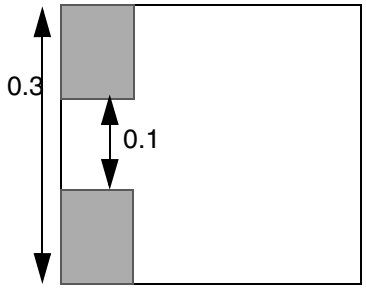
b) OK; has 0.01 and 0.04 overhang.



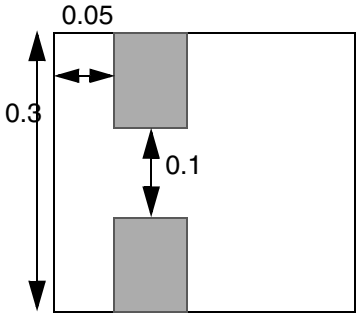
c) OK; meets wide-wire enclosure rule of 0.03 0.03.



d) OK; extra cut is <= 0.2 away; therefore, use min-width rule, and both cuts meet min-width enclosure rule of 0.0 and 0.5.



e) Violation. Extra cut is <= 0.2 away; therefore, use min-width rule, but cannot meet either 0.0 0.05 or 0.01 0.04 enclosure rules.



f) OK. Extra cut is <= 0.2 away; therefore use min-width rule, and both cuts meet the min-width enclosure rule of 0.0 0.05.

Example 4—minDualExtension with cutDistance and oaNoSharedEdge=true

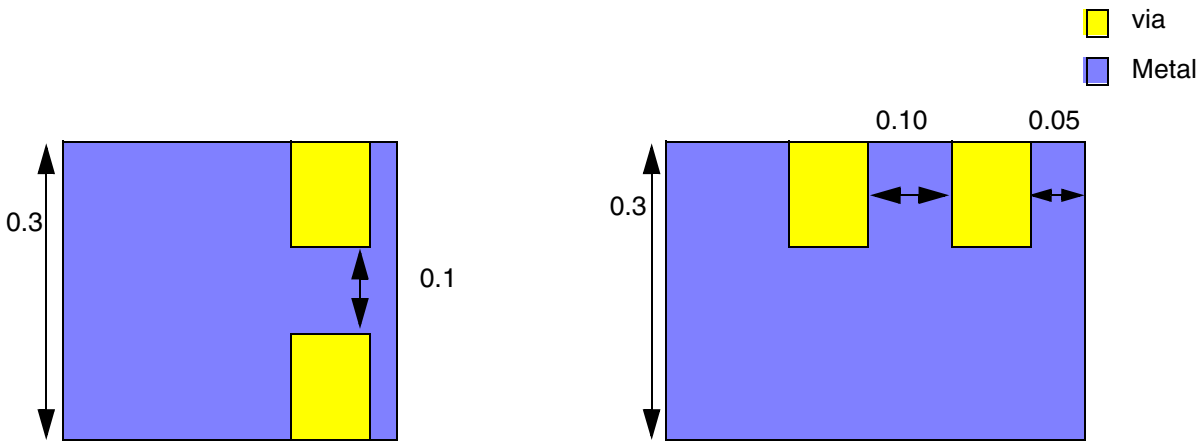
In this example, if the wire width is greater than or equal to 0.3 μm, then an enclosure of 0.03 μm in all directions is required unless there is a redundant cut within 0.2 μm. In this case, via cuts that share a wire edge are not considered redundant.

Format	Example

Tcl	`set_constraint_parameter -name oaNoSharedEdge -BoolValue true`
	`set_constraint_parameter -name cutDistance -Value 0.2`
	`set_layerpair_constraint -layer1 Metal1 -layer2 via1 \`
	`-constraint minDualExtension -hardness hard -row_name width \`
	`-OneDDualArrayTblValue {0.3 1 0.03 0.03}`
	Note: By default, `numCuts`=1.

LEF	`PROPERTY LEF58_ENCLOSURE`
	`"ENCLOSURE BELOW 0.03 0.03`
	`WIDTH 0.3 EXCEPTEXTRACUT 0.2 NOSHAREDEDGE ;" ;`

Illustration for minDualExtension with cutDistance and oaNoSharedEdge Example



a) Constraint passes and no checking is done because there is a redundant via < 0.2 μm `oaCutDistance` away that is not on the same edge and oaNoSharedEdge is true.

b) Violation. `oaNoSharedEdge=true` so the extra cut along the shared edge cannot be counted as a redundant via and the constraint applies, requiring 0.03 on all sides of cuts.

Example 5—minDualExtension with hasParallelRunLength Parameter

In this example, one of the following conditions must be satisfied:

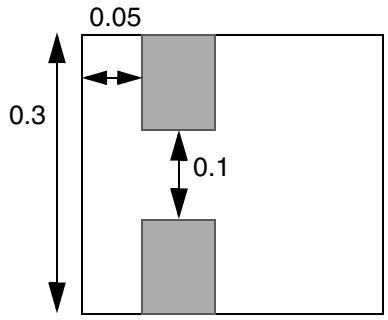
- A {0.03 0.03} enclosure applies for wires >= 0.3 μm wide except when there is a redundant via on the opposite edge within `cutDistance` that has a common parallel run length on > 0.
- Minimum width wires must have {0.0 0.05} or {0.02 0.02} enclosures.

Format	Example

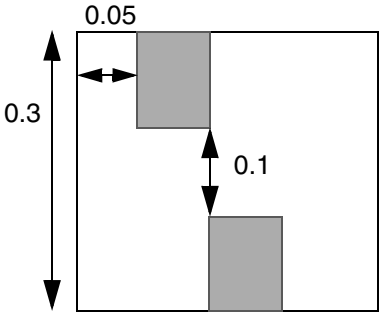
Tcl	`create_constraint_group -name Ext34 -opType or`
	`#minDualExtension settings for m3 layer below via34`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0  2 0.0  0.05  0.02 0.02 }`
	`set_constraint_parameter -name oaCutDistance -Value 0.2`
	`set_constraint_parameter -name hasParallelRunLength -BoolValue true`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0.3 1 0.03 0.03}`
	`#minDualExtension settings for m4 layer above via34`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0  2 0.0  0.05  0.02 0.02 }`
	`set_constraint_parameter -name oaCutDistance -Value 0.2`
	`set_constraint_parameter -name hasParallelRunLength -BoolValue true`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0.3 1 0.03 0.03}`

LEF	`PROPERTY LEF58_ENCLOSURE`
	`"ENCLOSURE 0.0  0.05 ;  #overhang    0.0  0.05`
	` ENCLOSURE 0.02 0.02 ;  #or overhang 0.02 0.02`
	` ENCLOSURE 0.03 0.03 WIDTH 0.3 EXCEPTEXTRACUT 0.2 PRL ;" ;`

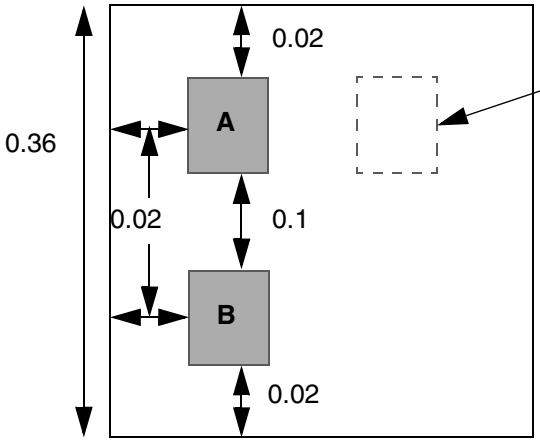
Illustrations for minDualExtension with hasParallelRunLength Example



a) OK. The extra cut is <= 0.2 away with parallel run length > 0, so use a min-width rule instead of the wide width rule of {0.03 0.03}. Both cuts meet the required {0.0 0.05} enclosure rule.



b) Violation. Extra cut is <= 0.2 away, but has no common parallel run length, so the wide width rule of {0.03 0.03} is required but not met for both cuts.



A cut here within `cutDistance` and with PRL > 0 to cut A would be a redundant cut, giving rule exception for cut A.

c) Violation. Wide wire enclosure of {0.03 0.03} is not satisfied on left edge of cut shapes A and B because there is no redundant cut shape on the opposite right edge.

Metal
Cut

Example 6— minDualExtension with length and width Parameters

This example defines a cut layer that requires an enclosure of 0.05 μm on opposite sides and 0.0 μm on the other two sides, as long as the total length enclosure on any two opposite sides is greater than or equal to 0.7 μm. Otherwise, it requires 0.05 μm on all sides if the total enclosure length is less than or equal to 0.7 μm. It also requires 0.10 μm on all sides if the

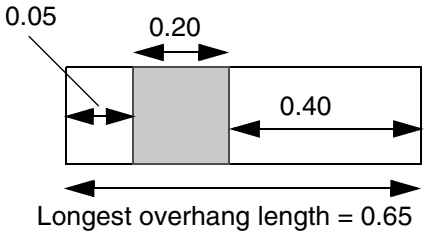
metal layer has a width that is greater than or equal to 1.0 µm. The following figure illustrates examples of violations and acceptable vias for the `minDualExtension` values)

Format	Example

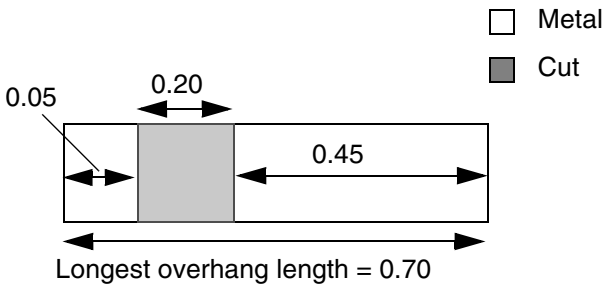
Tcl	`set_layer_constraint -constraint minWidth -layer via34 -Value 0.20`
	`set_layer_constraint -constraint minSpacing -layer via34 -Value 0.20`
	`create_constraint_group -name Ext34 -opType or`
	`# minDualExtension settings for m3 metal below via34 layer`
	`set_constraint_parameter -name length -Value 0.7`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -DualValue {0.05 0.0}`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0  1 0.05 0.05 \`
	`  1.0 1 0.1  0.1}`
	`# minDualExtension settings for m4 metal above via34 layer`
	`set_constraint_parameter -name length -Value 0.7`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -DualValue {0.05 0.0}`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	` -create true -group Ext34 \`
	` -constraint minDualExtension -OneDDualArrayTblValue { \`
	`  0  1 0.05 0.05 \`
	`  1.0 1 0.1  0.1}`

LEF	`LAYER via34`
	`TYPE CUT ;`
	`WIDTH 0.20 ;  #cuts .20 x .20 squares`
	`SPACING 0.20 ;  #via34 edge-to-edge spacing is 0.20`
	`ENCLOSURE 0.05 0.0 LENGTH 0.7 ; #overhang 0.05 0.0 if total >= 0.7`
	`ENCLOSURE 0.05 0.05 ;  #or, overhang 0.05 on all sides`
	`ENCLOSURE 0.10 0.10 WIDTH 1.0 ; #if width >= 1.0, always need 0.10`
	`...`
	`END via34`

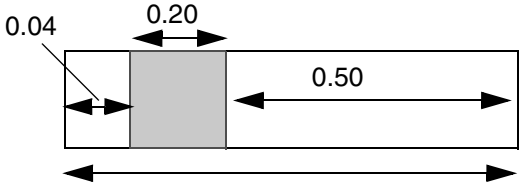
Illustrations for minDualExtension with Length and Width Example



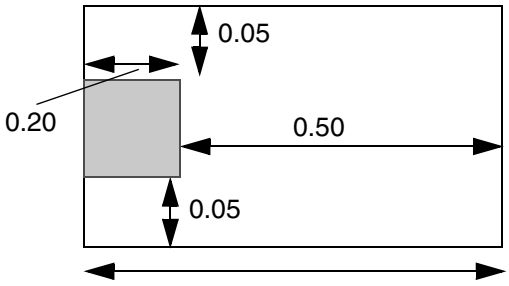
a) Violation. Longest overhang length < 0.70, and did not meet second rule of 0.05 on all sides.



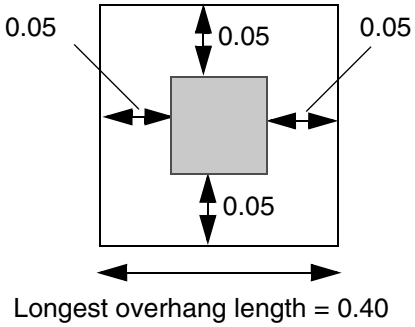
b) OK. Longest overhang length >= 0.70, and has 0.05 on opposite sides, and 0.0 on other sides.



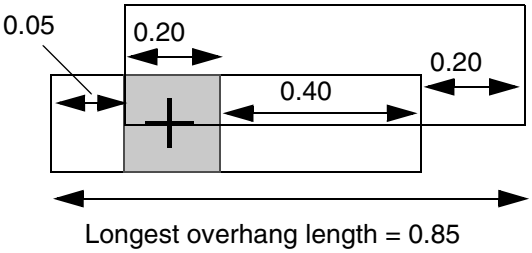
c) Violation. Longest overhang length >= 0.70, but does not have 0.05 on opposite sides, and did not meet second rule of 0.05 on all sides.



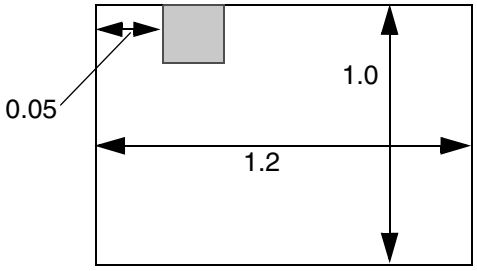
d) OK. Longest overhang length >= 0.70, and has 0.05 on opposite sides, and 0.0 on other sides.



e) OK. Total length < 0.7; therefore first rule fails, but second rule for 0.05 on all sides is met.



f) OK. Overhang length >= 0.70. (The center of the via cut is where the total overhang length is measured.)



g) Violation. Meets {0.05 0.0} rule, but width >= 1.0; therefore must meet third rule: 0.10 on all sides.

Example 7— minDualExtension with numCuts

In this example, VA vias with two or more same metal cuts must have 0.15 μm overhang on any two opposite sides of the routing layers. Otherwise, VA vias with only one cut must have an overhang of 0.20 μm on any two opposite sides of the routing layers.

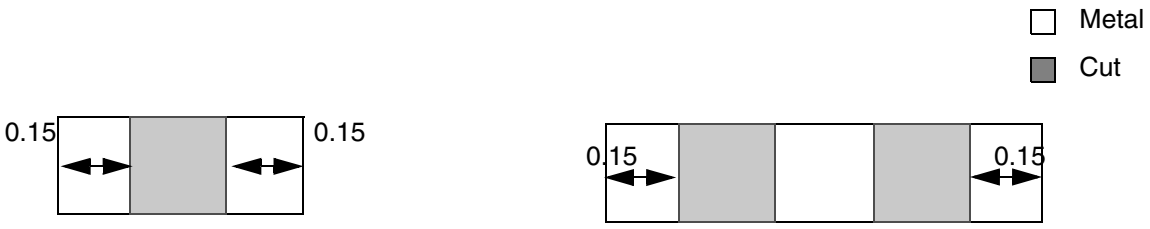
Format	Example

Tcl	`create_constraint_group -name Ext34 -opType and`
	`# minDualExtension settings for m3 metal below via34 layer`
	`set_constraint_parameter -name cutClass -DualValue {0.5 0.3}`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.20}`
	`# smaller extension required if there is an extra same metal cut`
	`set_constraint_parameter -name cutClass -DualValue {0.5 0.3}`
	`set_constraint_parameter -name numCuts -IntValue 2`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.15}`
	`# minDualExtension settings for m4 metal above via34 layer`
	`set_constraint_parameter -name cutClass -DualValue {0.5 0.3}`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.20}`
	`# smaller extension required if there is an extra same metal cut`
	`set_constraint_parameter -name cutClass -DualValue {0.5 0.3}`
	`set_constraint_parameter -name numCuts -IntValue 2`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.15}`

Format	Example

LEF	`PROPERTY LEF58_ENCLOSURE`
	`"ENCLOSURE CUTCLASS VA 0.000 0.20 ;`
	`ENCLOSURE CUTCLASS VA 0.000 0.15 EXTRACUT ; " ;`

Illustration for minDualExtension with numCuts Example



a) Violation. Only has one cut.

b) OK. Has at least 0.15 μm overhang on two opposite sides with 2 or more same metal cuts.

Example 8— minDualExtension with redundantCutDistance

In this example, if there is a redundant cut < 0.10 μm from a cut that has a {0.0 0.20} enclosure, then the redundant cut must have a {0.0 0.15} enclosure.

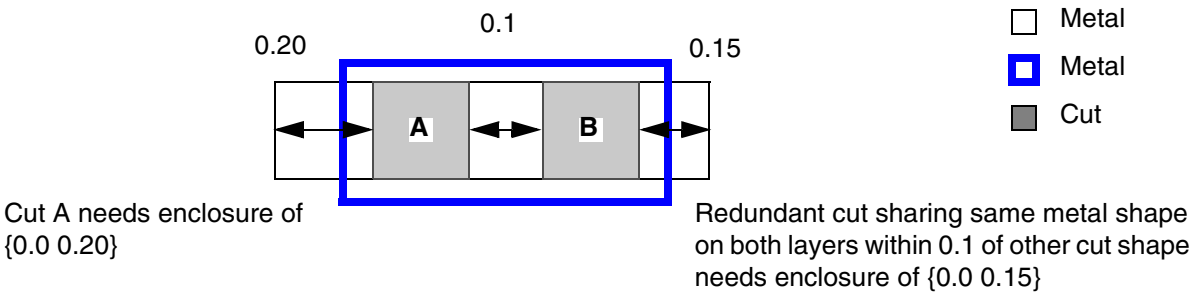
Format	Example

Tcl	`create_constraint_group -name Ext34 -opType and`
	`# minDualExtension settings for m3 metal below via34 layer`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.20}`
	`# specify the redundantCutDistance and enclosures`
	`set_constraint_parameter -name redundantCutDistance -Value 0.1`
	`set_layerpair_constraint -layer1 m3 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.15}`
	`# minDualExtension settings for m4 metal above via34 layer`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.20}`
	`# specify the redundantCutDistance and enclosures`
	`set_constraint_parameter -name redundantCutDistance -Value 0.1`
	`set_layerpair_constraint -layer1 m4 -layer2 via34 \`
	`-create true -group Ext34 \`
	`-constraint minDualExtension -DualValue {0.000 0.15}`

Format	Example

LEF	`PROPERTY LEF58_ENCLOSURE`
	`"ENCLOSURE 0.00 0.20 ;  #overhang  0.0  0.20`
	`ENCLOSURE 0.00 0.15 REDUNDANTCUT 0.1; " ;`

Illustration for minDualExtension with redundantCutDistance Example



Example 9— minDualExtension with exceptCutMetalEdgeExtension

In this example, if a metal shape of a cut via protrudes on a certain side of the wide wire, then the enclosure is not checked on that side. In addition, the enclosure based on the width of the protrusion must also be checked against the cut.

- For a cut in a wire of any width, the enclosure must be 0.00 μm on two opposite sides and 0.02 μm on the other two opposite sides.
- For a cut in wire with width greater than or equal to 0.15 μm and less than 0.3 μm, 0.02 μm overhang on four sides is needed.
- For a cut in wire with width greater than or equal to 0.3 μm, 0.03 μm overhang on four sides is needed.

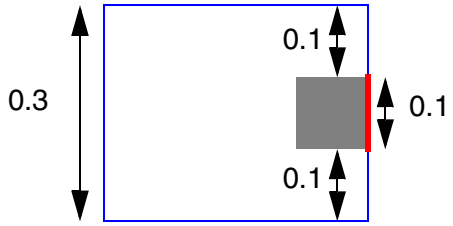
Format	Example

Tcl	`set_constraint_parameter -name exceptCutMetalEdgeExtension \`
	`-BoolValue true`
	`set_layerpair_constraint -layer1 v1 -layer2 m1 \`
	`-constraint minDualExtension \`
	`#  width  # ext1 ext2`
	`-OneDDualArrayTblValue {0  1 0.00 0.02`
	`0.15 1 0.02 0.02`
	`0.30 1 0.03 0.03}`

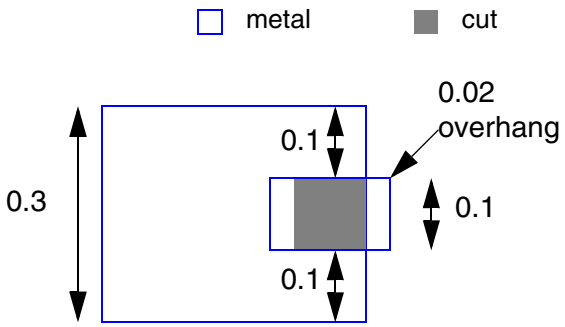
Format	Example

LEF	`PROPERTY LEF58_ENCLOSUREWIDTH`
	`  "ENCLOSUREWIDTH VIAOVERLAPONLY ;"`
	`ENCLOSURE 0.00 0.02 ;`
	`ENCLOSURE 0.02 0.02 WIDTH 0.15 ;`
	`ENCLOSURE 0.03 0.03 WIDTH 0.3  ;`

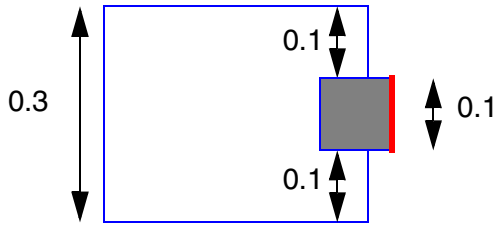
Illustration for minDualExtension with exceptCutMetalEdgeExtension Example



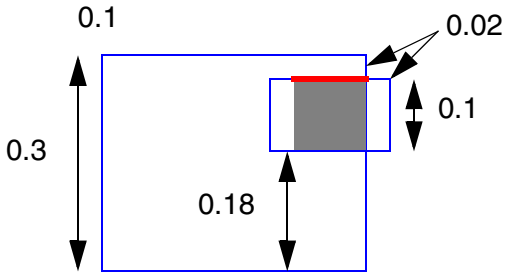
a) Violation, the right edge of the cut does not have 0.03 overhang



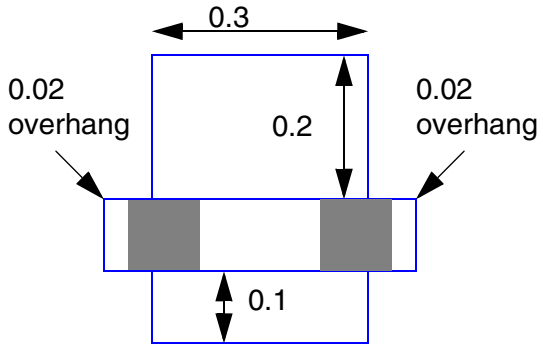
b) OK, the 0.02 protrusion would exempt the enclosure checking on the right edge, and the other three sides have enough overhangs



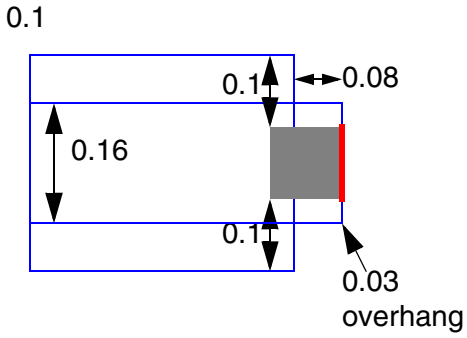
c) Violation, the right edge is exempted from the check because the metal of the cut is protruding, but the basic enclosure of {0.0 0.02} is not met.



d) Violation, the right edge is exempted by the protrusion, but the top edge still fails the 0.03 overhang required for the 0.3 metal.



e) OK, protrusion on right and left edges are exempt from checking, and the basic enclosure of {0.0 0.2} is met



f) Violation, the width of the protrusion is considered as the width of the max fractured rectangle of 0.16, and the corresponding enclosure of {0.02 0.02} is not met.

Example 10—minDualExtension with offCenterline and endSideOverhang

In this example, there must be short/end extensions of 0.05 μm and long side extensions of 0.03 μm for `Metal1` shapes beyond VxBAR cut class vias (0.02 x 0.04) on V1. The constraint applies only when the cut does not intersect with the centerline of the `Metal1` shape that is equal to 0.15 μm wide.

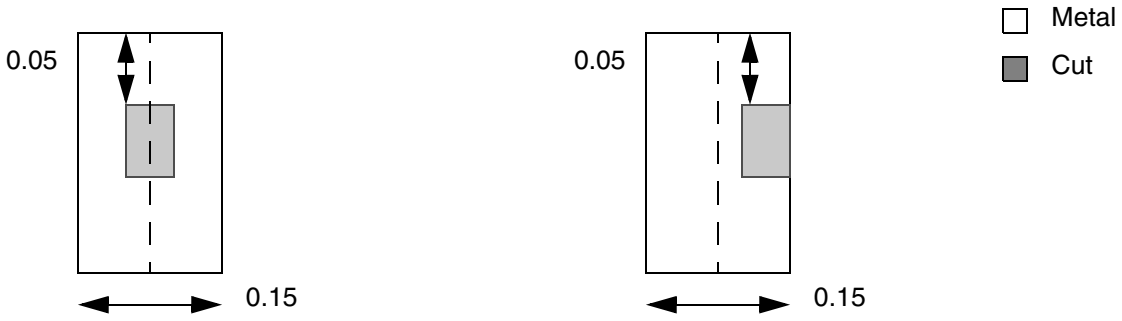
Format	Example

Tcl	`set_constraint_parameter -name offCenterline   -BoolValue true`
	`set_constraint_parameter -name endSideOverhang -BoolValue true`
	`set_constraint_parameter -name cutClass -DualValue { 0.02 0.04}`
	`set_layerpair_constraint -layer1 V1 -layer2 Metal1 \`
	`-constraint minDualExtension`
	`#                        width  # ext1 ext2`
	`-OneDDualArrayTblValue {0.15 1 0.05 0.03}`

LEF	`ENCLOSURE CUTCLASS VXBAR OFFCENTERLINE` `  END 0.05 SIDE 0.03 WIDTH 0.15`

Virtuoso	`spacingTables(`
	`(minOppExtension "V1" "Metal1"`
	`(( "width" nil nil )`
	`'offCenterline`
	`'cutclass (0.02 0.04)`
	`'endSide )`
	`( 0.15 ( (0.05 0.03) )`
	`) )`
	`) ;spacingTables`

Illustration for minDualExtension with offCenterline and endSideOverhang Example



a) OK. The via cut shape collides with the centerline of the layer1 shape which is the required width of 0.15 μm so the constraint does not apply.

b) Violation. The via cut shape does not collide with the centerline of the layer2 shape of width 0.15 and the required side enclosure of 0.03 is not met.

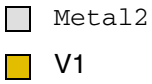
Example 11—minDualExtension with oaSpacingDirection and stepSizePair

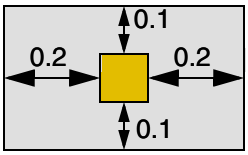
Specifies that the vertical extensions for `Metal2` shapes beyond V1 shapes must be greater than or equal to 0.1user units and the horizontal extensions must be greater than or equal to 0.2 user units. If greater than the minimum extension value, vertical extensions must be 0.1 plus an integer multiple of 0.04. The horizontal extensions have no step size restriction.

```
set_constraint_parameter -name oaSpacingDirection -IntValue 2
set_constraint_parameter -name stepSizepair -DualValue {0.04 0}
set_layerpair_constraint -constraint minDualExtension \
  -layer1 Metal2 -layer2 V1 -DualValue {0.1 0.2}
```

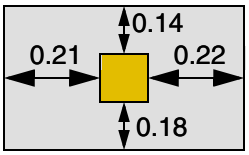
Example —minDualExtension with oaSpacingDirection and stepSizePair

`minDualExtension`	`{0.1 0.2}`
`oaSpacingDirection`	`2` (vertical)
`stepSizePair`	`{0.04 0}`

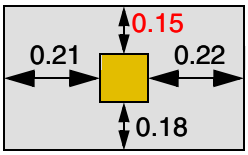




a) PASS. The constraint is met in both horizontal and vertical directions.



b) PASS. The constraint value is met in both directions. The vertical extensions use the `stepSizePair` formula 0.1+*n**0.04. The horizontal extensions are >= 0.2.



c) FAIL. The 0.15 vertical extension is not a `stepSizePair` multiple of 0.1+*n**0.04.

Related Topics

Extension Constraints

check_extension

minEndOfLineExtension

Sets the minimum extension of wire on a metal layer with respect to a cut layer when the wire has adjacent, parallel edges on three sides within the specified regions. If this constraint is not specified, or if it is specified but there are no parallel edges on three sides within the specified regions, a wire is extended beyond a cut shape dependent only on the minExtension constraint.

Important differences between minEndOfLineExtension and minExtension are:

- `minExtension` applies to a cut layer and sets the minimum extension for wires both above and below the cut layer.
- `minExtension` is not dependent on the end-of-line wire width.

minEndOfLineExtension Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

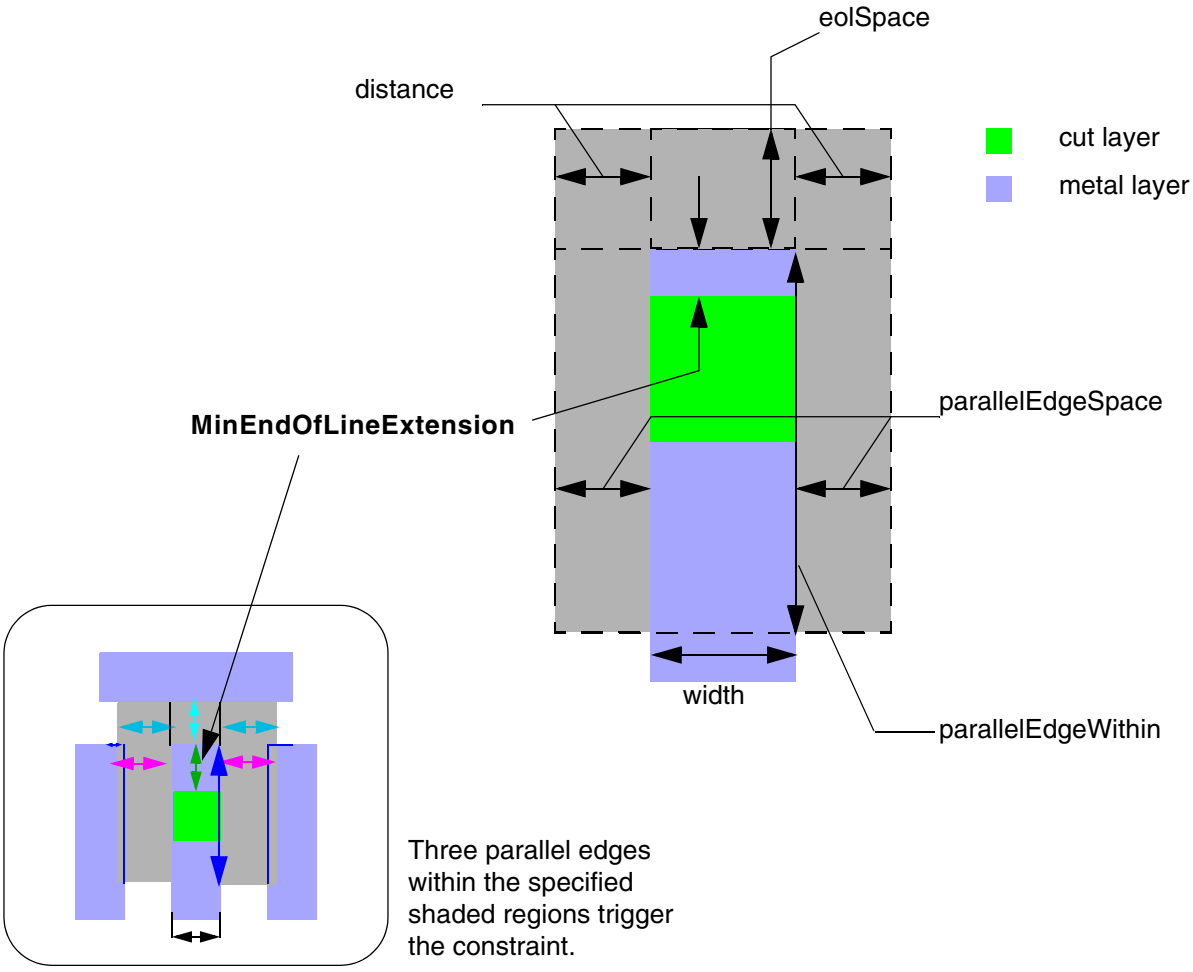
`minEndOfLineExtension` constraints have a `Value` that represents the minimum end-of-line extension that must be added when adjacent shapes satisfy the criteria defined by the five parameters.

Parameters

The `minEndOfLineExtension` constraint requires five parameters to specify when the constraint applies:

- `width` (`Value`) specifies the maximal width of the end-of-line wire (eolWidth). The constraint applies when the end-of-line wire width is less than this value.
- `eolSpace` (`Value`) specifies the end-of-line spacing. Combined with `distance`, the two parameters define the *lateral verification region*.

- `distance` (`Value`) specifies the verification distance for the end-of-line within distance. Combined with `eolSpace`, the two parameters define the *lateral verification region*. This constraint applies if there is a parallel edge in the lateral verification region and two parallel edges in the region defined by the `parallelEdgeSpace` and `parallelEdgeWithin` parameters.
- `parallelEdgeSpace` (`Value`) specifies the distance perpendicular to the end-of-line wire that parallel wires must lie within on both sides of the end-of-line wire for this constraint to apply.
- `parallelEdgeWithin` (`Value`) specifies the distance within which parallel wires are evaluated. If there are parallel wires within this distance on both sides of the end-of-line wire, the constraint applies.



Examples

Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.5`
	`set_constraint_parameter -name distance -Value 0.6`
	`set_constraint_parameter -name eolSpace -Value 0.7`
	`set_constraint_parameter -name parallelEdgeSpace -Value 0.6`
	`set_constraint_parameter -name parallelEdgeWithin -Value 1.2`
	`set_layerpair_constraint -constraint minEndOfLineExtension \`
	`-layer1 Metal1 -layer2 via1 -Value 0.2`

Related Topics

Extension Constraints

minEndOfLineEdgeExtension

Specifies an additional extension requirement when a cut shape lies near the end-of-a-line for the cut edge at the end of the line. The constraint does not apply to the end-of-line edge if it is greater than or equal to a given width. Width and cut class are required parameters for this constraint, and cut class must be a non-square cut class if the `longEdgeOnly` parameter is given.

- The extension may be a simple minimum extension, or optionally a smaller `exactExtension` may also be allowed.
- The constraint may not apply to edges with length less than *eolWidth* if those edges are not equal to the wire width (`equalRectWidth`).
- The constraint may also only apply to the extension of the end-of-line edge over the long/side edge of a non-square cut class (`longEdgeOnly`).

This constraint is not symmetric.

minEndOfLineEdgeExtension Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`minEndOfLineEdgeExtension` constraints have a `Value` that is the minimum extension of layer2 (metal) beyond layer1(cut).

Parameters

- `width` (`Value`, required) The constraint applies only to end-of-line edges with width less than this value, in user units.
- `cutClass` (`DualValue`, required) specifies the cut class for which this rule applies, given the cut class width and height.

- `equalRectWidth` (`BoolValue`, optional) specifies that the constraint only applies to end-of-line edges with length equal to the wire width.
- `exactExtension` (`Value`, optional) If specified, the overhang may match this value, instead of being greater than or equal to the constraint value. This parameter cannot be used with the `longEdgeOnly` parameter.
- `longEdgeOnly` (`BoolValue`, optional) If set and true, the constraint only applies to the long/side edges of a rectangular (non-square) cut at the end of the line. This parameter cannot be used with the `exactExtension` parameter.
- `maxRectEnclosureArea` (`AreaValue`, optional) If specified, the constraint only applies if the enclosure is rectangular and the area of enclosure is less than this value.
- `concaveCornerLength` (`Value`, optional) If specified, the constraint applies only if an edge adjoining an end-of-line edge that extends over the cut shape forms a concave corner with both concave edges less than this value in length.

Examples

Example 1—minEndOfLineEdgeExtension with longEdgeOnly

In this example, for end-of-line width <= 0.15, there must be a minimum 0.04 extension of `Metal1` past V1 at the long/side edge of a VB cut (0.06 wide and 1.0 in height) at the end of the line.

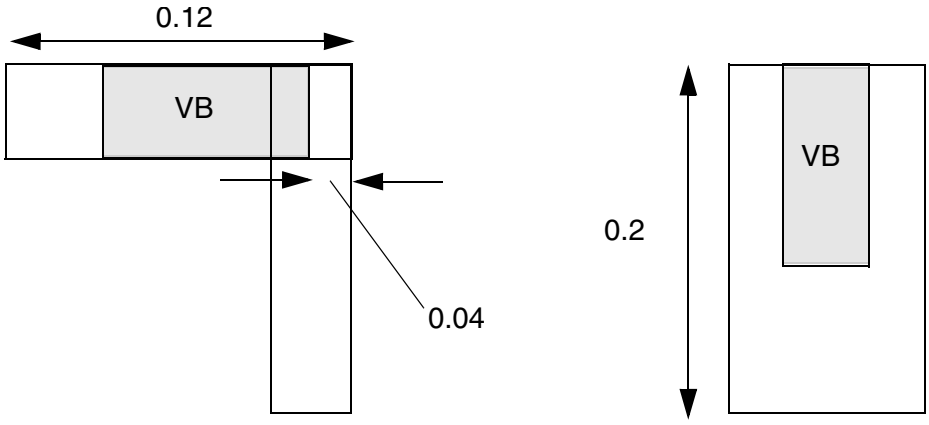
Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.15`
	`set_constraint_parameter -name cutClass -DualValue {0.06 0.1}`
	`set_constraint_parameter -name longEdgeOnly -BoolValue true`
	`set_layerpair_constraint -constraint minEndOfLineEdgeExtension \`
	`-layer1 V1 -layer2 Metal1 -Value 0.04`

LEF	`PROPERTY LEF58_EOLENCLOSURE`
	`"EOLENCLOSURE 0.15`
	`CUTCLASS VB`
	`LONGEDGEONLY 0.04 ; ";`

Virtuoso	`orderedSpacings(`
	`(minEndOfLineEdgeExtension V1 Metal1`
	`'cutClass (0.6 0.1)`
	`'width 0.15`
	`'longEdgeOnly`
	`0.04 )`
	`) ;orderedSpacings`

Illustration for minEndOfLineEdgeExtension with longEdgeOnly



a) Violation. 0.04 overhang must be on the side/long edge when the cut is on an EOL edge.

b) OK, the side/long edge of the cut is not on an EOL edge.

Example 2—minEndOfLineEdgeExtension with equalRectWidth and exactExtension

In this example, for end-of-line width <= 0.15, there must be a zero or minimum 0.05 extension of `Metal1` past V1 for square VA cuts (0.06 in width and height) at the end of the line. The constraint applies when the end-of-line edge has a length equal to the wire width.

Format	Example

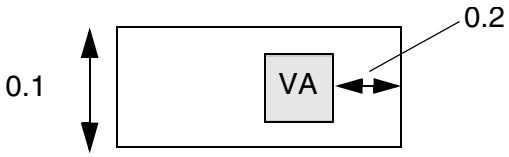
Tcl	`set_constraint_parameter -name width -Value 0.15`
	`set_constraint_parameter -name cutClass -DualValue {0.06 0.06}`
	`set_constraint_parameter -name exactExtension -Value 0.0`
	`set_constraint_parameter -name equalRectWidth true`
	`set_layerpair_constraint -constraint minEndOfLineEdgeExtension \`
	`-layer1 V1 -layer2 Metal1 -Value 0.05`

LEF	`PROPERTY LEF58_EOLENCLOSURE`
	`"EOLENCLOSURE 0.15 EQUALRECTWIDTH`
	`CUTCLASS VA 0.05 0.0 ; " ;`

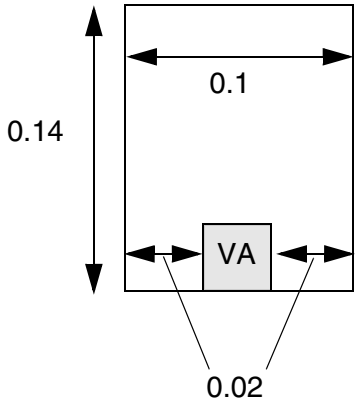
Virtuoso	`orderedSpacings(`
	`(minEndOfLineEdgeExtension V1 Metal1`
	`'cutClass (0.06 0.06)`
	`'width 0.15`
	`'equalRectWidth`
	`'exactExtension 0.0`
	`0.05 )`
	`) ;orderedSpacings`

Illustration for minEndOfLineEdgeExtension with equalRectWidth and exactExtension

a) OK. Zero overhang is allowed.



b) Violation. Overhang should be either 0.0 or >= 0.05.



c) OK, the vertical metal edges are not EOL edges with length equal to wire width (0.15), so the rule does not apply to them. The bottom edge has zero overhang, which is allowed (`exactExtension`). A violation would occur if `equalRectWidth` was not specified because the 0.02 overhang < the 0.05 required.

Example 3—minEndOfLineEdgeExtension with maxRectEnclosureArea

In this example, the value of minEndOfLineEdgeExtension is 0.025 under the following conditions:

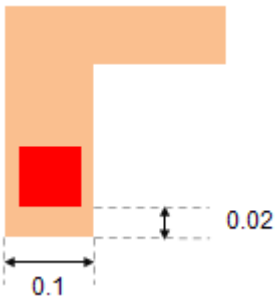
- The width is less than 0.2µm.
- The enclosure must be rectangular and the area of enclosure is less than 0.05µm.

Format	Example

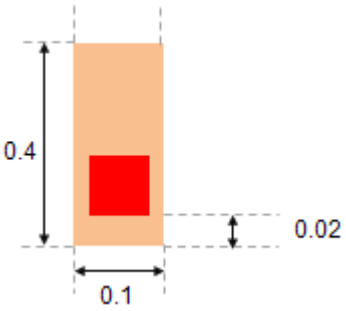
Tcl	`set_constraint_parameter -name width -Value 0.2`
	`set_constraint_parameter -name maxRectEnclosureArea -IntValue 0.05`
	`set_layerpair_constraint -constraint minEndOfLineEdgeExtension \`
	`-layer1 V1 -layer2 Metal1 -Value 0.025`

Illustration for minEndOfLineEdgeExtension with maxRectEnclosureArea

width = 0.2
maxRectEnclosureArea = 0.05
minEndOfLineEdgeExtension = 0.025



The constraint does not apply since the enclosure is non-rectangular.



The constraint applies because, a) the eolwidth (0.1) is less than width (0.2), b) the enclosure is rectangular with area (0.04), which is less than the maxRectEnclosureArea (0.05). However, there is a violation since the extension (0.02) is less than the constraint value (0.025).

Related Topics

Extension Constraints

minExtension

Specifies the minimum distance a shape on layer1 must extend past a shape on layer2. Extensions are specified from the outside edge of the shape on the second layer to the inside edge of the shape on the first layer. This constraint definition is not symmetric, implying that the minimum extension of layer1 past layer2 is not the same as the minimum extension of layer2 past layer1. The constraint applies only if parallel run length between the shapes is greater than zero.

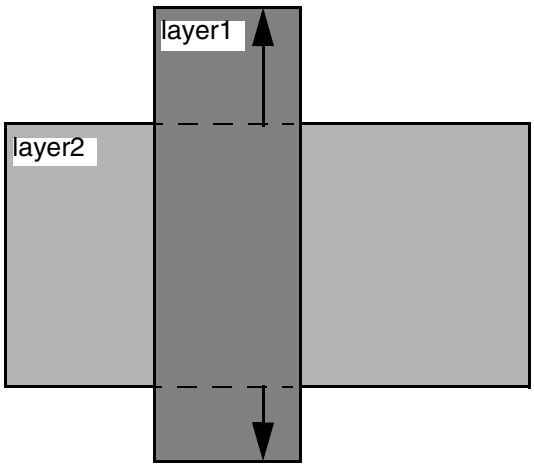
minExtension Quick Reference

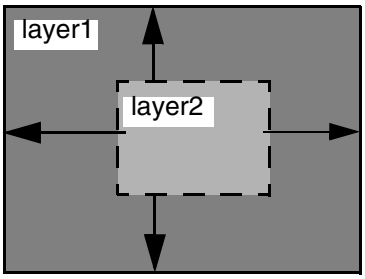
Constraint Type	Layer pair
Value Types	`Value`, `OneDTblValue`
Database Types	Design, Technology
Scope	net (for create_derived_vias), design, foundry
Category	Extension

Value Types

- `Value`

  Specifies the minimum extension in user units.

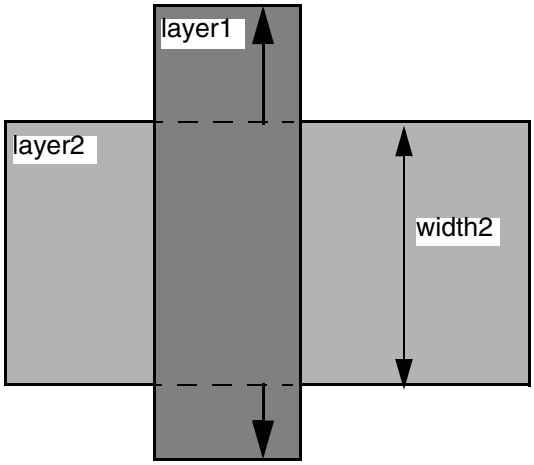




The extension of `layer1` over `layer2` must be greater than or equal to the value specified by this constraint.

- `OneDTblValue`

  Specifies the minimum spacing that changes according to the width of the wider of the two shapes. The width of the wider shape is the lookup key for the table. The table value is the minimum spacing. The width of a shape is defined as the smaller of the shape's two dimensions.



The extension of `layer1` over `layer2` must be greater than or equal to the value specified by this constraint, based on the width of the layer2 shape.

Parameters

- `coincidentAllowed` (`BoolValue`, optional) specifies whether shapes must meet the minimum extension or their edges can be coincident (`true`). By default and when set to `false`, the edges cannot be coincident.
- `cutClass` (`DualValue`, optional) specifies that the constraint applies for vias of the specified cut class.
- `oaSpacingDirection` (`IntValue`, optional) specifies that the constraint applies only to extensions measured in this direction. Valid values:
  0 any
  1 horizontal
  2 vertical
- `exceptEdgeLengthRange` (`RangeArrayValue`, optional) specifies that the constraint does not apply if the via edge length falls within the specified range.

Examples

Example 1—minExtension with a fixed value

This example sets the minimum extension of `Metal1` over `Metal2` to 2.5.

Format	Example
Tcl	`set_layerpair_constraint -constraint minExtension \ -layer1 Metal1 -layer2 Metal2 -Value 2.5`
Virtuoso	`orderedSpacings( (minExtension "Metal1" "Metal2" 2.5) )`

Example 2—minExtension 1D Table: Index width

This example sets the minimum extension of `Metal1` over `Metal2`, based on the width of the `Metal2` shape.

Format	Example
Tcl	`set_layerpair_constraint -constraint minExtension \ -layer1 Metal1 -layer2 Metal2 -row_name width \ -OneDTblValue {0.3 0.5 0.6 0.8}`

Related Topics

Extension Constraints

minExtensionEdge

Specifies the minimum extension of shapes on layer1 past shapes on layer2. Typically, layer1 is a metal layer and layer2 is a cut layer. The specified extension is typically larger than minExtension and minDualExtension values. The larger `minExtensionEdge` value is required only if the following conditions are met:

- The extension is only required if the metal enclosing the cut has width >= `width`, a parallel metal edge that is less than `parallelEdgeWithin` user units away with a parallel run length > `parallelEdgeLength`.
- The extension may be required only for cut shapes of particular cut classes, identified by their via width and height/length.
- For some processes, the rule does not apply if there is an extra cut and may require the extra cut to be less than or equal to `oaDistanceWithin` user units away.
- For some processes, the rule does not apply if there are two edges on either side of the metal shape that have parallel run length > `parallelEdgeLength` and within `parallelEdgeWithin`, parallel to the enclosing metal edge.

minExtensionEdge Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension
Group Operators	AND, OR

Value Type

`Value`	Represents the overhang of *`layer1`* past *`layer2`*, in user units.

Required Parameters

None

Optional Parameters

`cutClass`	Specifies that the constraint applies only to a cut class of the given dimensions. Type: `DualValue`
`twoSides`	Specifies that if the overhang on the layer above the cut layer of a via cut is less than or equal to the value of the constraint, the entire opposite edge must also have overhang less than or equal to the constraint value. **Note:** Cannot be specified with `width` or `minSpanLength`. Type: `BoolValue`
`endOfLineWidth`	Specifies that if this parameter is set and both overhangs are less than or equal to the value of the constraint, the overhang edges cannot be end-of-line edges with a length less than this parameter value. **Note:** Applies only if `twoSides` is true. Type: `Value`
`exceptConcaveCornerWithin`	Specifies that if this parameter is set and both overhangs are less than or equal to the value of the constraint, there must not be a concave corner on the above metal layer within this distance of the opposite overhang edges. **Note:** Applies only if `twoSides` is true. Type: `Value`
`cutToMetalSpacing`	Specifies that if this parameter is set and both overhangs are less than or equal to the value of the constraint, the distance between the cut edge to a different net wire on the above metal layer must be greater than or equal to this value. **Note:** Applies only if `twoSides` is true. Type: `Value`

`width`	Specifies that the constraint applies only if any edge of the cut layer is enclosed by metal that is greater than or equal to this width (*minWidth)*. **Note:** Cannot be specified with `twoSides` or `minSpanLength`. Type: `Value`
`parallelEdgeWidth`	Specifies that the constraint applies only if the parallel neighbor metal is greater than or equal to this value in width (*parWidth*). **Note:** Can only be used with `width`. Type: `Value`
`maxWidth`	Specifies that the width of the wire containing the via cut must be greater than or equal to `width` and less than this `maxWidth` value, and all the corresponding enclosure edge rules that meet the parallel and within conditions are applied to the cut. If any one of the `minExtensionEdge` constraints for a given set of layers specify `maxWidth`, then all of them must have `maxWidth`. **Note:** Can only be used with `width`. Type: `Value`
`minSpanLength`	Specifies that the width of the wire does not affect whether the constraint applies; instead, the enclosure edge rule applies if the span length of the wire containing the cut via in the direction perpendicular to the parallel run length is greater than or equal to this value. **Note:** Cannot be specified with the `width` or `twoSides` parameter. Type: `Value`

`maxSpanLength`	Specifies that the enclosure rule applies if the span length of the wire containing the cut via in the direction perpendicular to the parallel run length is greater than or equal to `minSpanLength` and less than this value. If any one of the `minExtensionEdge` constraints for a given set of layers specifies `maxSpanLength`, then all of them must have `maxSpanLength`. **Note:** Can only be used with `minSpanLength`. Type: `Value`
`parallelEdgeLength`	Specifies that the constraint applies when the parallel edge length (*parLength*) of the enclosing metal edge to another metal edge is greater than this value. **Note:** Required except if `twoSides` is specified. Cannot be used with `twoSides`. Type: `Value`
`parallelEdgeWithin`	Specifies that the constraint applies when the distance (*parWithin*) between a parallel metal edge and the enclosing metal edge is less than this value. **Note:** Required except if `twoSides` is specified. Cannot be used with `twoSides`. Type: `Value`
`parallelEdgeMinWithin`	Specifies that the constraint applies only if the parallel neighbor metal edge is greater than or equal to this distance away. **Note:** Applies only with `parallelEdgeWithin`. Type: `Value`
`exceptExtraCut`	Specifies that the constraint does not apply when there is another via cut in the same metal intersection, if set to `true`. **Note:** Cannot be used with `twoSides`. Type: `BoolValue`

`oaDistanceWithin`	Specifies that the constraint does not apply if `exceptExtraCut` is set to `true` and there is another via cut less than this distance from the first via cut (*extraCutWithin*). **Note:** Applies only with `exceptExtraCut`. Type: `Value`
`exceptTwoEdges`	Specifies that the constraint does not apply if the enclosing metal edges have parallel metal edges longer than `parallelEdgeLength` that are less than `parallelEdgeWithin` away on opposite sides. **Note:** Cannot be used with `twoSides`. Type: `BoolValue`
`exceptWithin`	Specifies that the `exceptTwoEdges` parameter applies only if the edges on opposite sides are greater than this value away. **Note:** Applies only when `exceptTwoEdges` is true. Type: `Value`
`includeCorner`	Specifies that the constraint applies to the corner of the cut, using Euclidian measurement. **Note:** Cannot be used with `twoSides`. Type: `BoolValue`

Examples

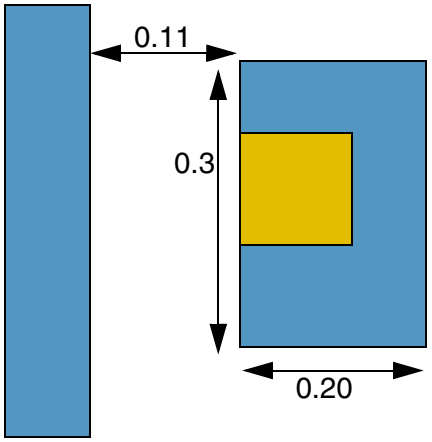
Example 1: minExtensionEdge

```
set_layerpair_constraint -constraint minDualExtension \
  -layer1 Metal1 -layer2 V1 -DualValue ( 0.0 0.05)

set_constraint_parameter -name width -Value 0.2
set_constraint_parameter -name parallelEdgeLength -Value 0.25
set_constraint_parameter -name parallelEdgeWithin -Value 0.11
set_layerpair_constraint -constraint minExtensionEdge \
  -layer1 Metal1 -layer2 V1 -Value 0.05
```

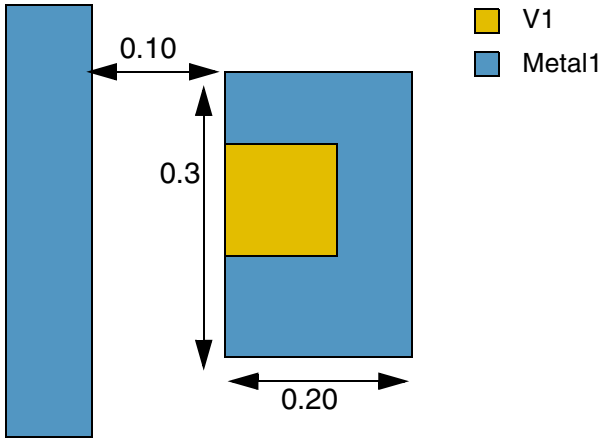
Requires an extension of 0.02 user units only if both of the following conditions are true:

- The `V1` via is enclosed by `Metal1` greater than or equal to 0.2 in width.

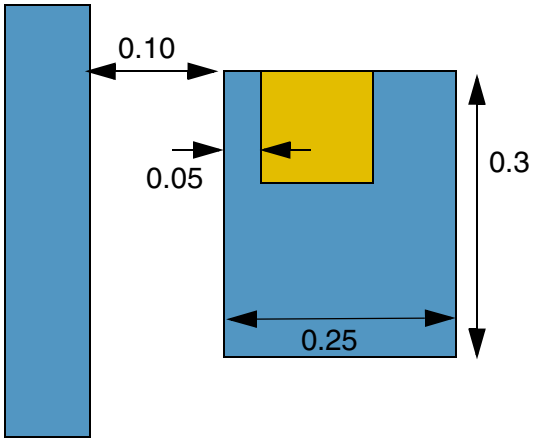
- There is a `Metal1` edge less than 0.11 user units away from the enclosing metal with a parallel run length greater than 0.25 user units.



a) PASS. Width >= 0.2, but spacing to neighbor is >= 0.11, so edge enclosure rule is not required. Cut meets the normal 0.00 0.05 enclosure rule.



b) FAIL. Width >= 0.2, spacing to neighbor is < 0.11, parallel run length is > 0.25, so edge enclosure rule of 0.02 is required but not met.



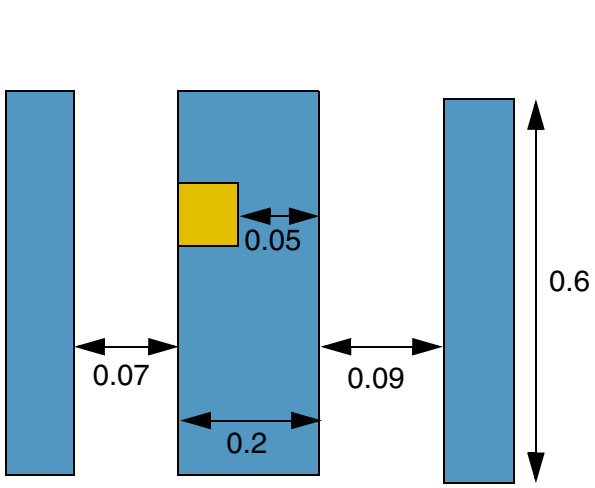
c) PASS. Width >= 0.2, spacing to neighbor is < 0.11, parallel run length is > 0.25, so edge enclosure rule of 0.02 is required and met. Cut also meets the normal 0.00 0.05 enclosure rule.

Example 2: minExtensionEdge with exceptTwoEdges

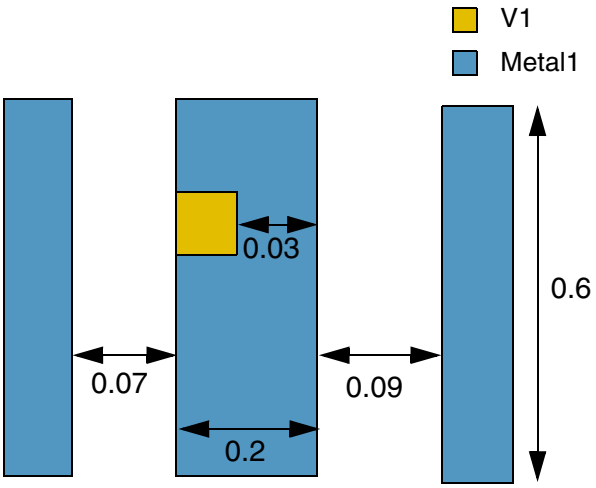
```
set_constraint_parameter -name width -Value 0.2
set_constraint_parameter -name parallelEdgeLength -Value 0.5
set_constraint_parameter -name parallelEdgeMinWithin -Value 0.08
set_constraint_parameter -name parallelEdgeWithin -Value 0.10
set_constraint_parameter -name exceptTwoEdges -BoolValue true
set_constraint_parameter -name exceptWithin -Value 0.12
set_layerpair_constraint -constraint minExtensionEdge \
  -layer1 Metal1 -layer2 V1 -Value 0.05
```

Requires an extension of 0.05 user units only if the `V1` via is enclosed by `Metal1` metal greater than or equal to 0.2 in width and there is a `Metal1` edge greater than or equal to 0.08

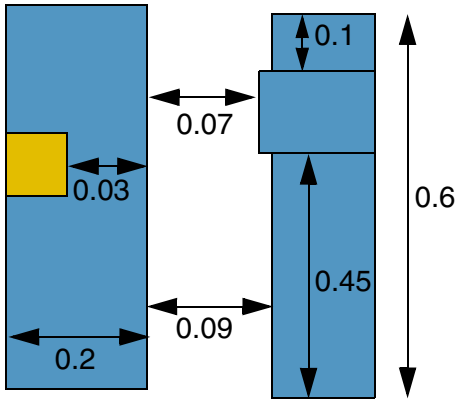
and less than 0.11 user units away with a parallel run length greater than 0.5 user units, **except** when the enclosing metal edges have parallel metal edges longer than `0.5` that are greater than `0.12` away on opposite sides.



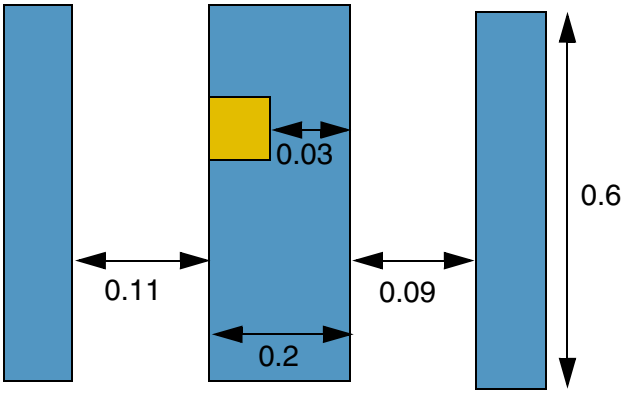
a) PASS. The left neighbor is 0.07 (< 0.08) away, and the rule does not apply. The right edge has 0.05 enclosure to fulfill the rule.



b) FAIL. The left neighbor is below the lower bound (< 0.08) and it is not a valid neighbor for two parallel neighbor edges exemption, so the constraint applies and 0.03 < 0.05 required extension.



c) PASS. The portion with 0.07 (< 0.08) spacing away is ignored, the rest having 0.45 and 0.1 (< 0.50) parallel run length should be treated individually, and the rule is not triggered.

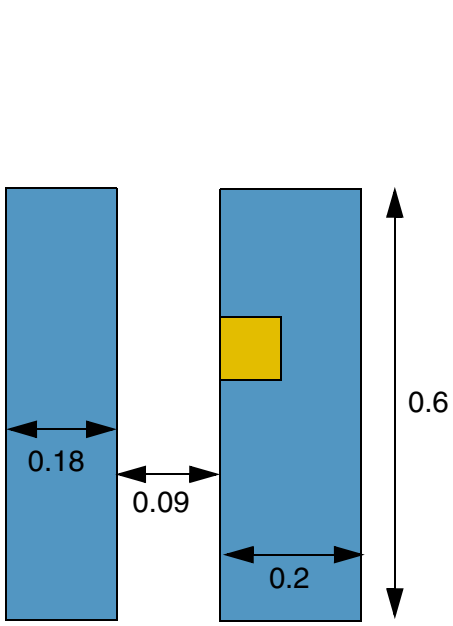


d) FAIL. There are two neighbor wires < 0.12 from the enclosing metal, so the rule does not apply. Violation if 0.12 `exceptWithin` is not specified with `exceptTwoEdges` because the left neighbor

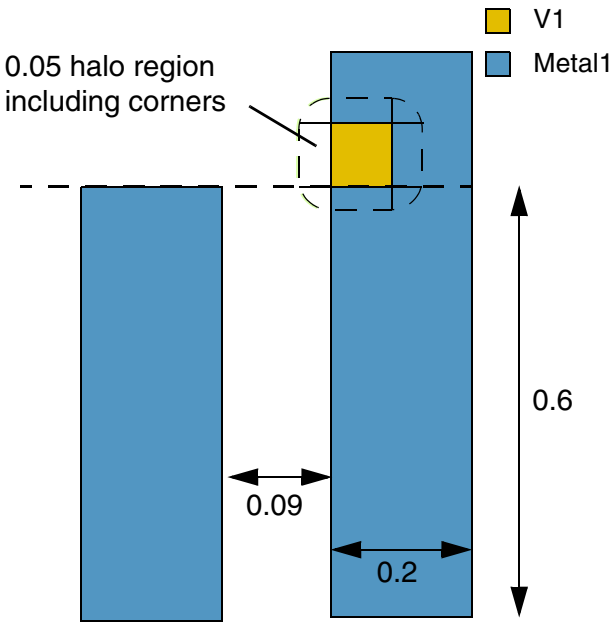
Example 3: minExtensionEdge with parallelEdgeWidth and includeCorner

```
set_constraint_parameter -name width -Value 0.2
set_constraint_parameter -name parallelEdgeLength -Value 0.5
set_constraint_parameter -name parallelEdgeWithin -Value 0.10
set_constraint_parameter -name parallelEdgeWidth  -Value 0.2
set_constraint_parameter -name includeCorner -BoolValue true
set_layerpair_constraint -constraint minExtensionEdge \
  -layer1 Metal1 -layer2 V1 -Value 0.05
```

Requires an extension of 0.05 user units, including corners, only if the `V2` via is enclosed by `Metal2` metal greater than or equal to 0.2 in width and there is a `Metal2` edge less than 0.1 user units away with a parallel run length greater than 0.5 user units and the parallel neighbor metal is greater than or equal to 0.2 in width.



a) OK. The width of the left neighbor wire is 0.18 < 0.20, which will not trigger the rule. Violation, if `parallelEdgeWidth` is not specified.

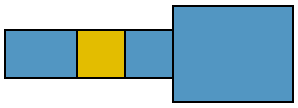


b) Violation. The bottom left corner overlaps with the projection of the neighbor wire. OK, if `includeCorner` is not specified.

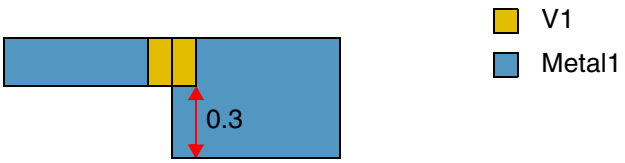
Example 4: minExtensionEdge with twoSides and endOfLineWidth

```
set_constraint_parameter -name endOfLineWidth -Value 0.15
set_constraint_parameter -name twoSides -BoolValue true
set_layerpair_constraint -constraint minExtensionEdge \
  -layer1 Metal1 -layer2 V1 -Value 0.02
```

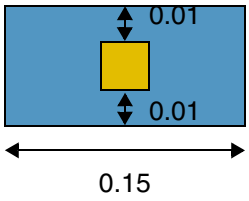
Specifies that if an overhang on the layer above the cut layer of a via is less than 0.02 user units, then the entire opposite edge must also have an overhang less than or equal to 0.02, and the overhang edges cannot be end-of-line edges less than 0.15 user units in length.



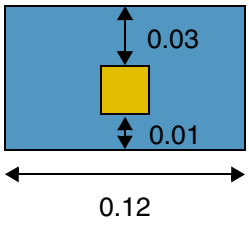
a) PASS. The Metal1 shape has an extension of 0 (<= 0.02) along the entire length of opposite edges of the V1 shape.



b) FAIL. The Metal1 extension on the bottom edge of the V1 shape is > 0.02 for part of the edge.



c) PASS. The Metal1 shape has an extension of 0.01 (<= 0.02) along the entire length of opposite edges of the V1 shape.

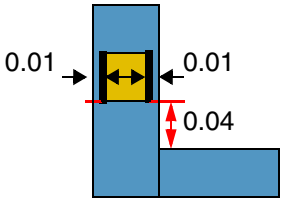


d) PASS. End-of-line edge of width 0.12 < endOfLineWidth (0.15) so the constraint does not apply.

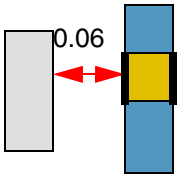
Example 5: minExtensionEdge with twoSides, exceptConcaveCornerWithin and cutToMetalSpacing

```
set_constraint_parameter -name exceptConcaveCornerWithin -Value 0.05
set_constraint_parameter -name twoSides -BoolValue true
set_constraint_parameter -name cutToMetalSpacing -Value 0.07
set_layerpair_constraint -constraint minExtensionEdge \
  -layer1 Metal1 -layer2 V1 -Value 0.02
```

Specifies that if an overhang on `Metal1` shape past a cut on `V1` is less than 0.02 user units, then the entire opposite edge must also have an overhang less than or equal to 0.02. Additionally, there must not be a concave corner on `Metal1` within 0.05 user units of those overhangs, and the spacing of a cut edge fulfilling the overhang requirement to a different net wire on `Metal2` must be at least 0.07.



a) FAIL. 0.01 extensions meet <= 0.02 overhang requirement but there is a concave corner < 0.05 from the right overhang edge.

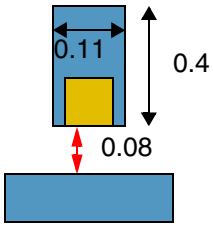


b) FAIL. Extensions meet 0.02 overhang requirement for `Metal1`, but spacing to `Metal2` of 0.06 < required 0.07 cutToMetalSpacing.

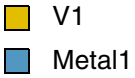
Example 6: minExtensionEdge with minSpanLength, parallelEdgeLength, and parallelEdgeWithin

```
set_constraint_parameter -name minSpanLength -Value 0.2
set_constraint_parameter -name parallelEdgeLength -Value 0.1
set_constraint_parameter -name parallelEdgeWithin -Value 0.09
set_layerpair_constraint -constraint minExtensionEdge \
  -layer1 Metal1 -layer2 V1 -Value 0.02
```

Specifies that the minimum enclosure edge depends on the span length of the `Metal1` wire containing the `V0` cut shape in the direction perpendicular to the parallel run length.



FAIL. Although the width of the `Metal1` shape is only 0.11, the span length is 0.4 (>0.2 `minSpanLength`) in the direction perpendicular to the edge containing the cut which has a parallel run length of 0.11 (> 0.1 `parallelEdgeLength`) with another `Metal1` shape. The rule is triggered but the spacing to the parallel `Metal1` shape is 0.08, which less than the required 0.9 `parallelEdgeWithin`.



Related Topics

Extension Constraints

check_extension

minExtensionOnLongSide

Specifies the minimum extension for a layer1 shape past the long side edge of a layer2 shape. This constraint may not apply to all layer1 or layer2 widths. Two parameters indicate the set of width ranges for each layer when the constraint applies. This constraint applies to rectangular shapes only. Non-rectangular shapes are ignored. This constraint is not symmetric.

minExtensionOnLongSide Quick Reference

Constraint Type	Layer pair
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

Value	Represents the minimum long side extension, in user units.

Optional Parameters

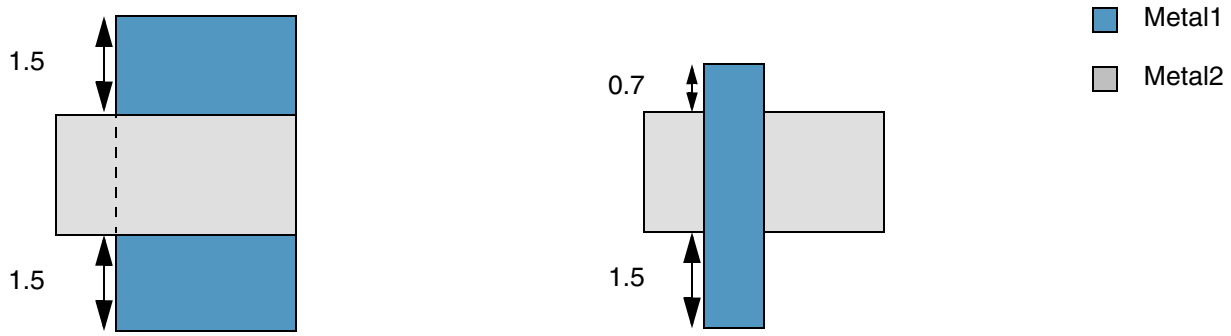
widthRangeArray	Specifies that the constraint applies only if the layer1 width is in one of the ranges. Type: RangeArrayValue
otherWidthRangeArray	Specifies that the constraint applies only if the layer2 width is in one of the ranges. Type: RangeArrayValue

Examples

The following example specifies that Metal1 shapes must extend at least 0.15 past the long side edge of Metal2 shapes.

```
set_layerpair_constraint -constraint minExtensionOnLongSide \
  -layer1 Metal1 -layer2 Metal2 -Value 1.5
```

Illustration for minExtensionOnLongSide



a) OK, `Metal1` extensions of 1.5 past long side edges of `Metal2` meets requirement.

b) Violation. `Metal1` extensions >= 1.5 required on both long side edges of `Metal2`.

Related Topics

Extension Constraints

check_extension

minExtensionToCorner

Restricts how close a cut shape can be to the corners of the enclosing metal. The corners can be either convex or concave. For convex corners, the rule is defined by forming a triangular keep-out region from the convex corner where cuts are not allowed. The convex corner check does not apply to corners that form end-of-line edges, as defined by the `eolWidth` parameter. The first layer specified is the metal layer and the second is the cut layer. This constraint must be satisfied in addition to existing extension rules.

minExtensionToCorner Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`Value`	Specifies in user units the required extension of layer1 over layer2. See the descriptions of the `toConvexCorner` and `toConcaveCorner` parameters for the interpretation of this value.

Required Parameters

`cutClass`	Specifies that the constraint only applies to cut shapes of the specified cut class. Type: `DualValue`

Optional Parameters

`extendCornerBy`	Specifies that the check is applied not only to the corner, but also to the edge-forming corners for a length up to that specified by this parameter. Type: `Value`

`exceptCornerTouch`	Specifies that the constraint does not apply if the cut touches the corner. The default is false. Type: `BoolValue`
`viaEdgeTypeParam`	Specifies that the constraint applies only to via edges of the specified type. Type: `IntValue`
`enclosure`	Specifies that if the enclosure is less than the first value to the convex length edge, the adjacent cut edges must have an enclosure larger than the second given value. Measured in user units. Type: `DualValue`
`convexLength`	Specifies that the concave corner must have an edge that is less than this length and that connects to a convex corner on the other end to trigger the rule. Type: `Value`
`parallelExtension`	Specifies the extension past the cut to a concave corner. If the width of an edge forming the concave corner <= the `width` parameter, the length of the adjacent edge is >= the `length` parameter, then form a search window by extending the extension on the concave corner along both sides of the edge with width <= the width parameter. The extension stops if it does not have a common projection run length with the opposite edge. Any cut edge parallel to the metal edge with width <= the `width` parameter inside that search window must have an extension >= the `parallelExtension` parameter. Type: `Value`
`length`	See `parallelExtension`. Type: `Value`
`width`	See `parallelExtension`. Type: `Value`

`toConcaveCorner`	Specifies that the constraint applies between cut shapes on layer2 and concave corners of the enclosing metal shapes on layer1. The constraint can be thought of as a distance from the exterior edge of the cut shapes on layer2 to concave corners on layer1. Type: `BoolValue`
`minLength`	Specifies the minimum length of the end-of-line edge on both sides or else it is not considered to be an EOL length. The constraint then only applies to convex corners that do not touch an edge of length strictly less than `minLength`. Type: `Value`
`toConvexCorner`	Specifies that the constraint value defines a triangular keep-out region in each convex corner of shapes on layer1. Shapes on the layer2 which overlap this region violate the constraint. The triangular keepout region is defined either by the smaller of the edge length or the constraints value. Type: `BoolValue`
`endOfLineWidth`	The triangular keepout region cannot touch an end-of-line edge with width strictly less than the value of `endOfLineWidth`, meaning that the constraint does not apply near such corners. This parameter must be specified if `toConvexCorner` is specified. Type: `Value`

Related Topics

Extension Constraints

minInnerVertexProximityExtension

Specifies the minimum extension for a cut that is near an inner vertex of the metal shape that encloses it.

minInnerVertexProximityExtension

Constraint Type	Layer pair
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

Value	Specifies in user units the required extension for cuts that are positioned near the inner vertex of an enclosing metal layer.

Required Parameters

width	Specifies that the constraint applies only if the width of the enclosing metal shape is less than or equal to this value. Type: Value
jogHeight	Specifies that the constraint applies only when the height of the jog that forms the inner vertex is greater than this value. Type: Value

Optional Parameters

cutClass	Specifies that the constraint applies only to cuts of the specified width and length. Type: DualValue

`withinRange`	Specifies that if the cut is within this range of the inner vertex, then it must meet the extension specified by constraint value. Type: `RangeValue`
`otherExtension`	Specifies that the constraint applies only if the extension on the opposite side of the inner vertex is less than this value. Type: `Value`

Related Topics

Extension Constraints

minNeighborExtension

Specifies the minimum extension of a shape on `layer1` past a shape on `layer2` on opposite pairs of sides, when the `layer2` cut shape has a neighbor wire within less than the specified parallel within distance (`parallelEdgeWithinRange`), and a common parallel run length (`parallelEdgeLengthRange`) with the `layer2` cut shape greater than or equal to the specified parallel run length *on only one side*.

Note: If the parallel run length is a negative value, then the distance between the `layer2` cut shape and the neighbor wire should be less than the absolute value of the specified parallel run length.

In addition, a second parallel within distance and parallel run length may be specified. The second parallel run length must be less than the first parallel run length and the second parallel within distance must be greater than the first parallel within distance. In this case, the constraint would be violated if there is a neighboring wire in the parallel within range (within within2) having a common parallel run length to the cut either less than the first parallel run length and greater than or equal to the second parallel run length on any one or both sides.

Also, the extensions required for wide wires are usually larger, but the `minNeighborExtension` can be even larger. Unless a cut is completely inside a wide wire, the `minNeighborExtension` constraint should also be checked. The `minNeighborExtension` constraint can optionally apply only when the `layer1` wire containing the `layer2` cut is less than a given width (`maxWidth`).

Optionally, the minimum neighbor extension only applies if the extension of the shape on the metal layer below is less than the specified other extension value (`otherExtension`) on either side perpendicular to the side of the `layer2` cut having neighbors or the wire direction containing the cut on the metal layer above.

minNeighborExtension Quick Reference

Constraint Type	Layer pair (Symmetric: no)
Value Types	DualValue
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Parameters

- `cutClass` (DualValue, required) specifies that the constraint applies for the given cut class width and height on `layer2`.
- `parallelEdgeLengthRange` (RangeValue, required) specifies the upper and, optionally, the lower bound for the parallel run length between the cut and the neighbor wire. If the parallel run length is a negative value, then the distance between the `layer2` cut shape and the neighboring wire must be less than the absolute value of the specified parallel run length.
- `parallelEdgeWithinRange` (RangeValue, required) specifies the lower and, optionally, the upper bound for the within distance.
- `otherExtension` (Value, optional) specifies that the constraint applies only if the extension on the below metal layer is less than this value on either side perpendicular to the side having neighbors or the wire direction containing the cut on the above metal layer. To use a metal layer other than the below metal layer for this parameter, set the `layer` parameter.
- `layer` (LayerValue, optional) specifies the layer on which the `otherExtension` parameter applies.
- `width` (Value, optional) specifies that the constraint applies when the width of the wire containing the cut is less than this value.

Examples

Example—minNeighborExtension with One parallelEdgeLengthRange and parallelEdgeWithinRange

In this example, if there is a `Metal2` neighbor wire less than 0.11um (`parallelEdgeWithinRange`) from a `V2` cut with a common parallel run length greater

than or equal to 0.08um (`parallelEdgeLengthRange`) on only one side of the cut, and the cut is on a `Metal2` wire that is less than 0.1um in width (`width`), then the cut must have `Metal2` extensions greater than or equal to 0.09um on all four sides. Otherwise, one of the following must be satisfied:

- `Metal2` extensions past `V2` cuts must be greater than or equal to 0.05um on two opposite sides, with zero extension required on the two other sides.
- If the width of the `Metal2` wire containing the `V2` cut is greater than or equal to 0.1um, then the `Metal2` extensions must be greater than or equal to 0.06um on two opposite sides, with zero extension required on the two other sides.

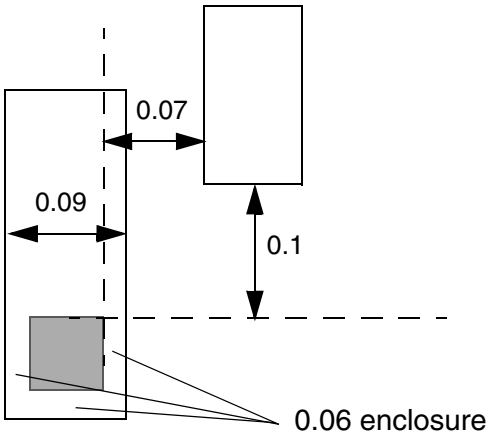
Format	Example

Tcl	`create_constraint_group -name m2_v2 -optype or -db design`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	` -constraint` **`minDualExtension`** `-DualValue {0.05 0.00} \`
	` -create true -group m2_v2`
	`set_constraint_parameter -name width -Value 0.1`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	` -constraint` **`minDualExtension`** `-DualValue {0.06 0.00} \`
	` -create true -group m2_v2`
	`set_constraint_parameter -name width -Value 0.1 \`
	`set_constraint_parameter -name parallelEdgeLengthRange \`
	` -RangeValue { "-0.11"}`
	`set_constraint_parameter -name parallelEdgeWithinRange \`
	` -RangeValue { 0.08 }`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	` -constraint` **`minNeighborExtension`** `-DualValue {0.09 0.09}`

techfile	`spacingTables(`
	`  (`**`minOppExtension`** `"Metal2" "V2"`
	`  (("width" nil nil))`
	`  (`
	`  0.0 (0.05 0.0)`
	`  0.1 (0.06 0.0)`
	`  )`
	`  )`
	`) ;spacingTables`
	`orderedSpacings(`
	`  (`**`minNeighborExtension`** `"Metal2" "V2"`
	`  'paraLengthRange -0.11`
	`  'withinRange 0.08`
	`  'width 0.1`
	`  (0.09 0.09)`
	`  )`
	`); orderedSpacings`

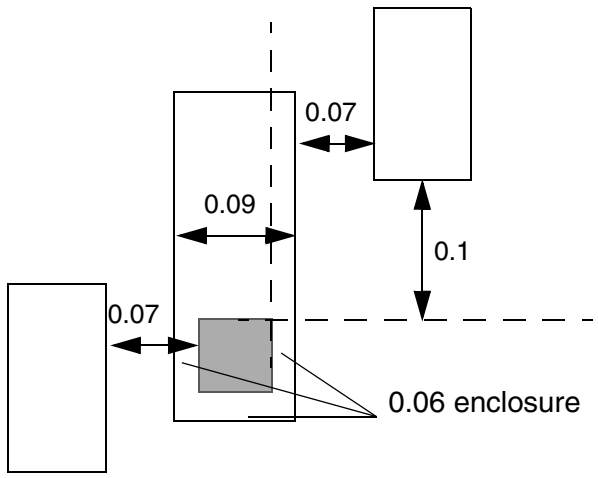
Illustration of—minNeighborExtension with One parallelEdgeLengthRange and parallelEdgeWithinRange

`minDualExtension` {0.05 0.00}
`minDualExtension` {0.06 0.00} `width` 0.1

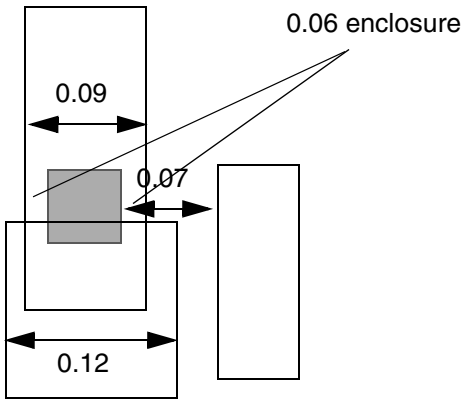


a) Violation. The neighbor wire is within the required `parallelEdgeWithinRange` (0.07 < 0.08) of the cut and the parallel run length is 0.1 < 0.11, so the `minNeighborExtension` constraint applies and is not met (0.06 < 0.09) on three sides.

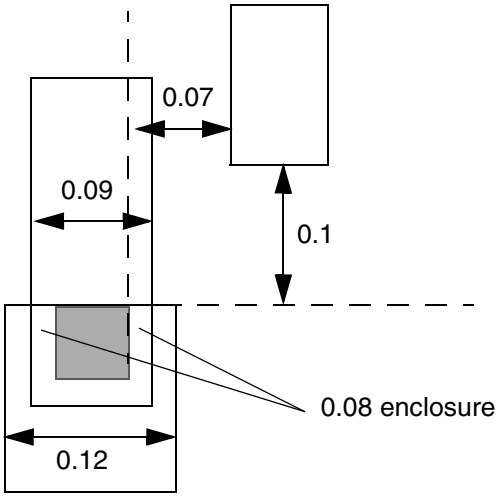
`minNeighborExtension` {0.09 0.09}
`parallelEdgeLengthRange` -0.11
`parallelEdgeWithinRange` 0.08
`width` 0.1



b) OK. The `minNeighborExtension` constraint does not apply because there are **two** neighbor wires within the within range and with the required parallel run length, each on a different side of the cut. The `minDualExtension` constraint of {0.05 0.00} applies and is fulfilled.



c) Violation. The cut is not fully enclosed in the wide wire, so the `minNeighborExtension` constraint applies and is not met on two sides of the cut (0.06 < 0.09).



d) OK, the cut is completely inside the 0.12 wide wire, and the `Metal2` extensions meet the `minDualExtension` {0.06 0.00} requirement.

Example—minNeighborExtension with Two parallelEdgeLengthRange and parallelEdgeWithinRange

In this example, one of the following must be satisfied:

- If there is a `Metal2` neighbor wire less than 0.04um from a `V2` 0.03x0.03 cut with a common parallel run length greater than or equal to 0.08um *on only one side* of the cut, then the cut must have `Metal2` extensions greater than or equal to 0.015um on all four sides.
- `Metal2` extensions past `V2` 0.03x0.03 cuts must be greater than or equal to 0.02um on two opposite sides, with zero extension required on the two other sides.

In addition, it is only legal to have `Metal2` neighbor wires on both sides that are less than 0.04um from the cut with common parallel run length to the cut less than 0.08um (within the small search window in the following figure, or, no neighbor on either side that is less than 0.06um from the cut with common parallel run length less than 0.1um (within the large search window in the following figure.

Format	Example

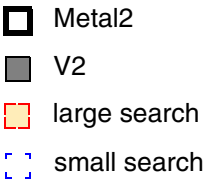
Tcl	`set_constraint_parameter -name cutClass -DualValue {0.03 0.03}`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	`  -constraint` **`minDualExtension`** `-DualValue {0.02 0.00} \`
	`  -create true -group m2_v2`
	
	`set_constraint_parameter -name cutClass -DualValue {0.03 0.03}`
	`set_constraint_parameter -name parallelEdgeLengthRange \`
	`  -RangeValue { "[-0.08 -0.1]"}`
	`set_constraint_parameter -name parallelEdgeWithinRange \`
	`  -RangeValue { "[0.04 0.06]"}`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	`  -constraint` **`minNeighborExtension`** `-DualValue {0.015 0.015}`

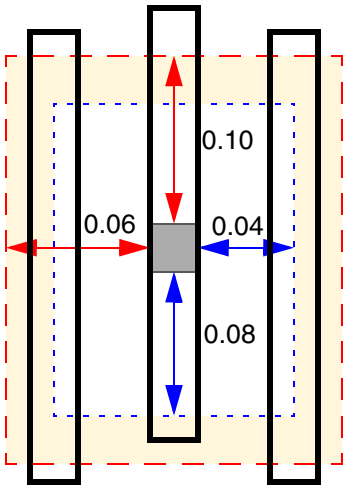
techfile	`orderedSpacings(`
	`  (`**`minOppExtension`** `"Metal2" "V2"`
	`  'cutClass 0.03 0.03`
	`  (0.02 0.0)`
	`  )`
	`) ;spacingTables`
	
	`orderedSpacings(`
	`  (`**`minNeighborExtension`** `"Metal2" "V2"`
	`  'cutClass 0.03 0.03`
	`  'paraLengthRange "[-0.08 -0.1]"`
	`  'withinRange "[0.04 0.06]"`
	`  (0.015 0.015)`
	`  )`
	`); orderedSpacings`

Illustration of minNeighborExtension with Two parallelEdgeLengthRange and parallelEdgeWithinRange

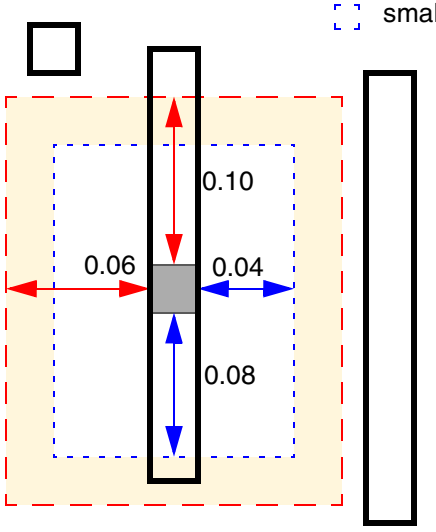
```
minDualExtension {0.02 0.0}
  cutClass {0.03 0.03}
```

```
minNeighborExtension {0.015 0.015}
  parallelEdgeLengthRange "[-0.08 -0.1]"
  parallelEdgeWithinRange "[0.04 0.06]"
  cutClass {0.03 0.03}
```

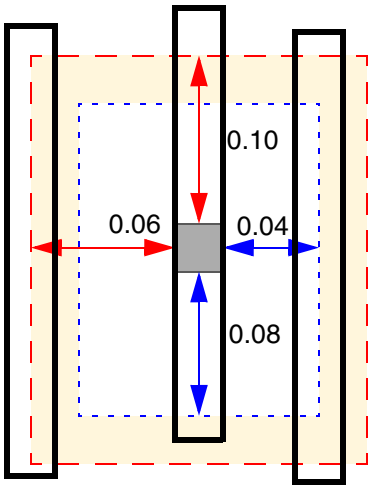




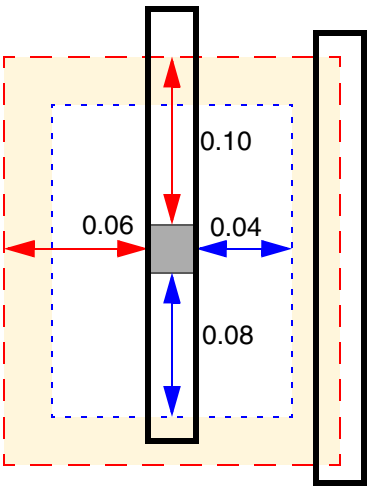
a) OK. It is legal to have two neighbor wires within the small search window on both sides. The `minNeighborExtension` does not apply but the `minDualExtension` of {0.02 0.0} applies and is met.



b) OK. It is legal to have no neighbor wires within the larger search window. The `minNeighborExtension` does not apply but the `minDualExtension` of {0.02 0.0} applies and is met.



c) Violation. The cut with only one neighbor wire within the smaller search window must have extensions >= 0.015 on all four sides.



d) Violation. It is not legal to have just one neighbor wire between the small and large search window.

Example—minNeighborExtension with Two parallelEdgeLengthRange and parallelEdgeWithinRange and otherExtension

In this example, if a `V2` 0.03x0.03 cut has `Metal1` extensions less than 0.05um on either side perpendicular to either the side having `Metal2` neighbors or the `Metal2` wire containing the cut, then the following applies:

- If there is a `Metal2` neighbor wire less than 0.04um from a `V2` 0.03x0.03 cut with a common parallel run length greater than or equal to 0.08um *on only one side* of the cut, then the cut must have `Metal2` extensions greater than or equal to 0.015um on all four sides.
- It is only legal to have `Metal2` neighbor wires on both sides that are less than 0.04um from the cut with common parallel run length to the cut less than 0.08um, or, no neighbor on either side that is less than 0.06um from the cut with common parallel run length less than 0.1um.

Otherwise, `Metal2` extensions past `V2` 0.03x0.03 cuts must be greater than or equal to 0.02um on two opposite sides, with zero extension required on the two other sides.

Format	Example

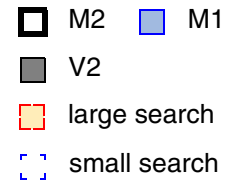
Tcl	`set_constraint_parameter -name cutClass -DualValue {0.03 0.03}`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	`  -constraint` **`minDualExtension`** `-DualValue {0.02 0.00} \`
	`  -create true -group m2_v2`
	
	`set_constraint_parameter -name cutClass -DualValue {0.03 0.03}`
	`set_constraint_parameter -name parallelEdgeLengthRange \`
	`  -RangeValue { "[-0.08 -0.1]"}`
	`set_constraint_parameter -name parallelEdgeWithinRange \`
	`  -RangeValue { "[0.04 0.06]"}`
	`set_constraint_parameter -name otherExtension -Value 0.05`
	`set_layerpair_constraint -layer1 Metal2 -layer2 V2 \`
	`  -constraint` **`minNeighborExtension`** `-DualValue {0.015 0.015}`

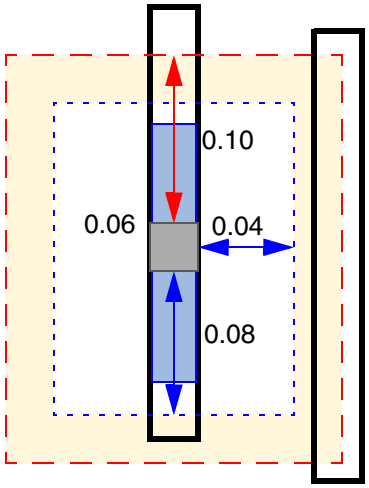
techfile	`orderedSpacings(`
	`  (`**`minOppExtension`** `"Metal2" "V2"`
	`  'cutClass 0.03 0.03`
	`  (0.02 0.0)`
	`  )`
	`) ;spacingTables`
	
	`orderedSpacings(`
	`  (`**`minNeighborExtension`** `"Metal2" "V2"`
	`  'cutClass 0.03 0.03`
	`  'otherExtension 0.05`
	`  'paraLengthRange "[-0.08 -0.1]"`
	`  'withinRange "[0.04 0.06]"`
	`  (0.015 0.015)`
	`  )`
	`); orderedSpacings`

Illustration of minNeighborExtension with Two parallelEdgeLengthRange and parallelEdgeWithinRange and otherExtension

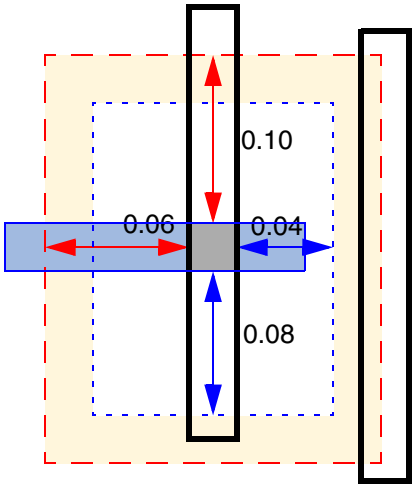
```
minDualExtension {0.02 0.0}
  cutClass {0.03 0.03}
```

```
minNeighborExtension {0.015 0.015}
  parallelEdgeLengthRange "[-0.08 -0.1]"
  parallelEdgeWithinRange "[0.04 0.06]"
  otherExtension 0.05
  cutClass {0.03 0.03}
```

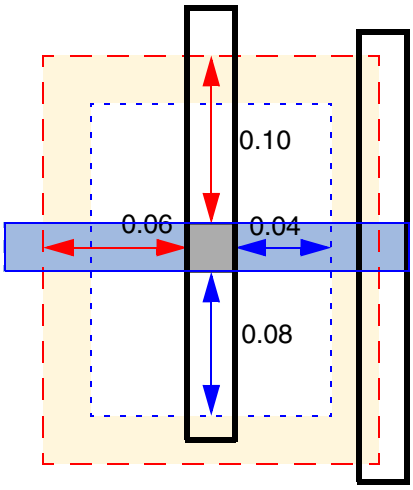




a) Violation. The `minNeighborExtension` constraint is triggered because the below metal `Metal1` extension is 0.0 < 0.05 on both sides perpendicular to the above `Metal2` wire direction. It is not legal to have a `Metal2` parallel wire within the large search window.



b) Violation. The `minNeighborExtension` constraint is triggered because the below metal `Metal1` extension is < 0.05 on the right side of the cut perpendicular to the above `Metal2` wire direction. It is not legal to have a `Metal2` parallel wire within the large search window.



c) OK. The `minNeighborExtension` constraint is not triggered because the below metal `Metal1` extension is >= 0.05 on both sides perpendicular to the above `Metal2` wire direction. The {0.02 0.0} `minDualExtension` applies and is met.

Related Topics

Extension Constraints

minQuadrupleExtension

Specifies the minimum extension of a shape on `layer1` with a given minimum width past a shape on `layer2`. The extension of a shape on `layer1` can have up to four different values specified to apply to the four sides of the `layer2` shape, which is typically a square or rectangle. The first two extensions apply to one pair of opposite edges (for example, top and bottom, or left and right), and the other two extensions apply to the remaining pair of opposite edges.

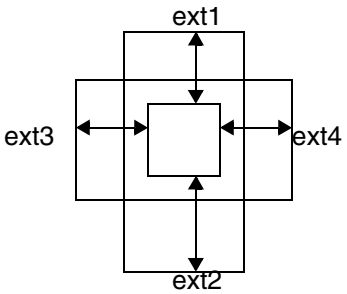
Usually, an OR constraint group should contain only settings of the same constraint with different parameters. If any of the settings is satisfied, the constraint requirement is met. The exception is that minQuadrupleExtension, minDualExtension, and minWireExtension can be bound together in an OR group and if any of the constraints is satisfied, the extension requirement is met.

minQuadrupleExtension Quick Reference

Constraint Type	Layer pair (non- symmetric)
Value Types	`OneDDualArrayTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Types

`minQuadrupleExtension` constraints have a `OneDDualArrayTblValue` which is indexed by wire width, in user units. Each row contains a minimum width, the number of extension pairs associated with the width, and the extension `DualValue` pairs. Each DualValue pair specifies an extension quadruple. The first DualValue specifies the first two extension values on opposite sides, and the next DualValue specifies the remaining two extension values for the other two opposite sides.



Opposite side extensions are paired with this notation:
{{ext1 ext2} {ext3 ext4}}
Braces are optional.

For example,

```
#                      width #pairs 1st pair     2nd ext pair
-OneDDualArrayTblValue {0.3 2      {0.05 0.05}  {0.08 0.1}}
```

For a wire greater than or equal to 0.3 user units in width, a via touching or overlapping the wire must have minimum extensions of {0.05 0.05} on opposite sides, and {0.08 0.1} on the other two opposite sides.

Parameters

- `cutClass` (`DualValue`, optional) specifies that the constraint only applies to cut shapes of the specified cut class.
- `allSides` (`BoolValue`, optional) If true and if any of the extension sets are asymmetric (extension1 is not equal to extension2, or extension3 is not equal to extension4), then the minimum extension on all four sides of the `layer2` shape must be greater than or equal to the smallest of the four specified extensions, including the corners. In this case, the extensions for the minimum width `layer1` wire shape, which touches or overlaps the `layer2` cut shape are used.

  If the parameter is false, then the extensions apply to the maximum width `layer1` wire shape that the `layer2` cut shape overlaps or touches.
- `extensionType` (`OneDDualArrayTblValue`, optional) specifies special attributes for each set of four extension values in the same table format as the constraint value. All four `extensionType` integer values for an extension set should be the same. Valid values are given in the following table.

extensionType Parameter Values

String	Integer Value	Description
`None`	0	Normal extension
`minSum`	1	Allows two opposite sides to have any extension, as long as the sum of the two extensions is greater than or equal to the sum of the specified extensions and the smaller extension is greater than or equal to the smaller one of the specified extensions.

Examples

In this example,

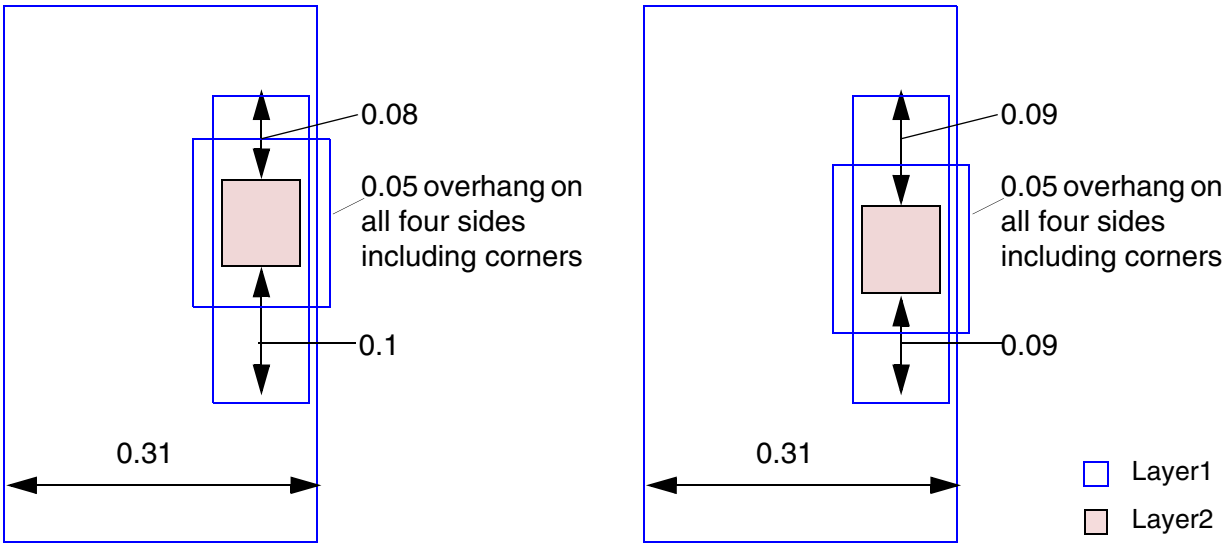
- Extension pairs for layer1 wires with minimum width of 0.3 past layer2 cut shapes are {0.05 0.05} and {0.08 0.1}.
- If any of the extension sets are asymmetric (extension1 is not equal to extension2, or extension3 is not equal to extension4), then the minimum extension on all four sides of the `layer2` shape must be greater than or equal 0.05.
- Allows two opposite sides to have any extension, as long as the sum of the two extensions is greater than or equal to the sum of the specified extensions and the smaller extension is greater than or equal to the smaller one of the specified extensions.

Format	Example

Tcl	`set_constraint_parameter -name allSides true`
	`set_constraint_parameter -name extensionType \`
	`  -OneDDualArrayTblValue { 0.3 4 1 1 1 1 0 0 0 0 } \`
	`  -row_interpolation snap_up \`
	`  -row_extrapolation {snap_up snap_down} \`
	`  -col_interpolation snap_down \`
	`  -col_extrapolation {snap_down snap_down}`
	`set_layer_constraint -constraint minQuadrupleExtension \`
	`  -layer1 Metal1 -layer2 Via1 \`
	`  -OneDDualArrayTblValue { 0.3 4 0.05 0.05 0.08 0.1} \`
	`  -row_interpolation snap_up \`
	`  -row_extrapolation {snap_up snap_down} \`
	`  -col_interpolation snap_down \`
	`  -col_extrapolation {snap_down snap_down}`

Virtuoso	`spacingTables(`
	`  ( minQuadrupleExtension Metal1 Via1`
	`  (( "width" nil nil ) 'allSides)`
	`  (`
	`  0.3 ((0.05 0.05 0.08 0.1 'minSum))`
	`  )`
	`) ;spacingTables`

Illustration for minQuadrupleExtension



a) OK. Layer1 wire width of 0.31 > 0.3 for asymmetric extensions has 0.05 extension on all four sides, including corners, and the top and bottom edges satisfy 0.08 and 0.1 extensions.

b) OK. The sum of the top and bottom extensions (0.09+0.09=0.18) is equal to (0.08+0.1=0.18) and the smallest extension of 0.09 is greater than the smaller specified extension value of 0.08.

Related Topics

Extension Constraints

minSideExtension

Specifies the extension that either the short or long side edge of a layer1 shape must extend past a layer2 shape.

Optionally, the constraint may apply only if the layer2 extension is less than or equal to a certain value.

This constraint may not apply to all layer1 or layer2 widths. Two parameters indicate the set of width ranges for each layer when the constraint applies.

This constraint applies to rectangular shapes only. Non-rectangular shapes are ignored.

This constraint is not symmetric.

minSideExtension Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`minSideExtension` constraints have a `Value` that represents the minimum side extension in user units.

Parameters

- `otherExtension` (`Value`, optional) specifies that the constraint applies only if the layer2 extension past layer1 is less than or equal to this value.
- `longSideEdge` (`BoolValue`, optional) When `true`, the constraint applies to the long side edge of layer1, otherwise (false) applies only to the short side edge.
- `widthRangeArray` (`RangeArrayValue`, optional) specifies that the constraint applies only if the layer1 width is in one of the ranges.
- `otherWidthRangeArray` (`RangeArrayValue`, optional) specifies that the constraint applies only if the layer2 width is in one of the ranges.

- `exceptEdgeLengthRangeArray` (`RangeArrayValue`, optional) specifies that the constraint does not apply if the layer2 edge length is in this range.

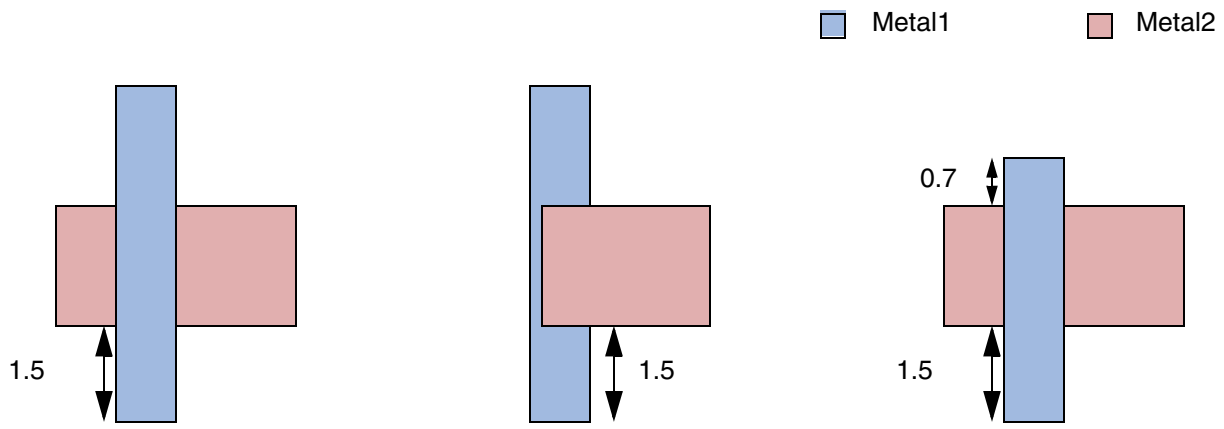
Examples

Example 1—minSideExtension With No Parameters

In this example, the short edge of `Metal1` shapes must extend at least 1.5 μm past `Metal2` shapes.

Format	Example
Tcl	`set_layerpair_constraint -constraint minSideExtension \ -layer1 Metal1 -layer2 Metal2 -Value 1.5`
Virtuoso	`orderedSpacings( (minSideExtension "Metal1" "Metal2" 1.5) ) ;orderSpacings`

Illustration for minSideExtension With No Parameters



a) OK, extension met.

b) OK, extension met.

c) Violation, extension >= 0.15 μm required on both sides

Example 2—minSideExtension With otherExtension Parameter

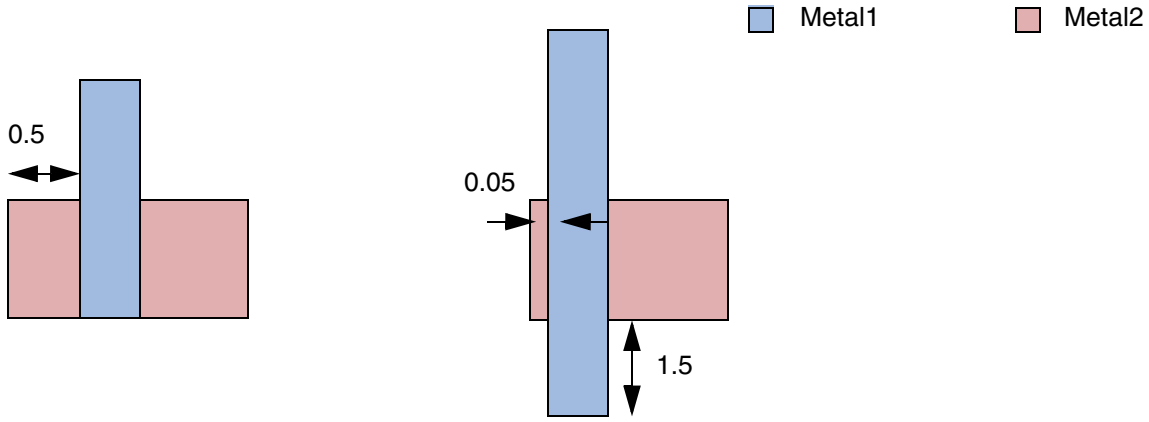
In this example, short edge of `Metal1` shapes must extend past `Metal2` shapes at least 0.2 µm when `Metal2` shapes extend past `Metal1` shapes by less than or equal to 2.0 µm.

Format	Example

Tcl	`set_constraint_parameter -name otherExtension -Value 0.2`
	`set_layerpair_constraint -constraint minSideExtension \`
	`-layer1 Metal1 -layer2 Metal2 -Value 2.0`

Virtuoso	`orderedSpacings(`
	`(minSideExtension "Metal1" "Metal2" 'otherExtension 0.2 2.0)`
	`) ;orderSpacings`

Illustration for minSideExtension With otherExtension Parameter



a) OK, constraint does not apply because `Metal2` extension > 0.2

b) Violation. `Metal1` extension of 2.0 is needed because `Metal2` extension of 0.05 is

Example 3—minSideExtension With longSideEdge Parameter

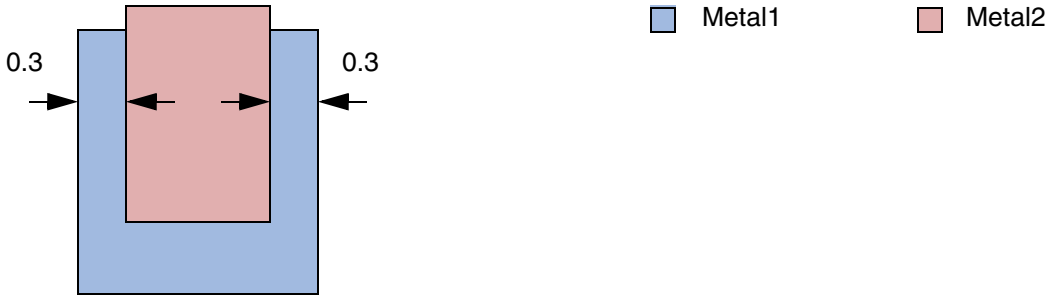
In this example, the long edge of `Metal1` shapes must extend at least 0.15 μm past `Metal2` shapes.

Format	Example

Tcl	`set_constraint_parameter -name longSideEdge -BoolValue true`
	`set_layerpair_constraint -constraint minSideExtension \`
	`-layer1 Metal1 -layer2 Metal2 -Value 0.15`

Virtuoso	`orderedSpacings(`
	`(minSideExtension "Metal1" "Metal2" 'longEdges true 0.15)`
	`) ;orderSpacings`

Illustration for minSideExtension With longSideEdge Parameter



a) OK, the long-side edge of layer1 extends more than 0.15 past layer2.

Example 4—minSideExtension With exceptEdgeLengthRangeArray and longSideEdge Parameters

In this example, the extension of a long edge of a layer1 shape past a layer2 shape must be greater than or equal to 0.12um, except when the layer2 edge is less than or equal to 0.2um, equal to 0.5um, or equal to 0.6um.

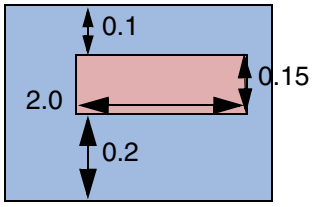
Format	Example

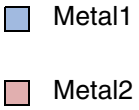
Tcl	`set_constraint_parameter  -name longSideEdge -BoolValue true`
	`set_constraint_parameter -name exceptEdgeLengthRangeArray -RangeArrayValue {"<=0.2" "0.5" "0.6"}`
	`set_layerpair_constraint -layer1 layer1 -layer2 layer2 -constraint minSideExtension -Value 0.12`

techfile	`orderedSpacings(`
	`  (minSideExtension "layer1" "layer2"`
	`   'longEdge`
	`   'exceptEdgeLengthRanges ("<=0.2" "0.5" "0.6")`
	`   0.12`
	`  )`
	`); orderedSpacings`

Example of minSideExtension With exceptEdgeLengthRangeArray and longSideEdge Parameters

```
minSideExtension              0.12
  longSideEdge                true
  exceptEdgeLengthRangeArray {"<=0.2" "0.5" "0.6"}
```





a) The constraint only applies to the top and bottom edges of the layer1 shape (`longSideEdge`). The constraint applies to the top and bottom edges of the layer2 shape because 2.0 is not in any of the `exceptEdgeLengthRangeArray` parameter ranges.
Violation, the top edge extension of 0.1 is less than the 0.12 constraint value.
OK, the bottom edge extension of 0.2 meets the 0.12 constraint value.

Related Topics

Extension Constraints

check_extensions

minTouchingDirectionExtension

Specifies an enclosure for the field end of the gate that is different from the enclosure for the channel.

minTouchingDirectionExtension Quick Reference

Constraint Type	Layer array
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`minTouchingDirectionExtension` constraints have a `Value`.

Parameter

- `distanceMeasureType`

  (`IntValue`, optional) This parameter specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1).

Related Topics

Extension Constraints

Euclidean and Manhattan Spacing Constraints.

minViaJointExtension

Some processes require an extension for joint edges that is different from the extension to other edges. The `minViaJointExtension` constraint specifies the minimum extension from a cut on layer1 to two consecutive layer2 joint edges, and can optionally apply to a specific cut class.

The first layer (layer1) should be the cut layer and the second layer (layer2) should be an interconnect or routing layer.

This constraint is not symmetric.

minViaJointExtension Quick Reference

Constraint Type	Layer pair
Value Types	`Value`, DualValue
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Types

`minViaJointExtension` constraints have the following value types:

- `Value`

  Represents the required extension, in user units, of the interconnect or routing layer shape past the cut shape to one of the joint edges.

- `DualValue`

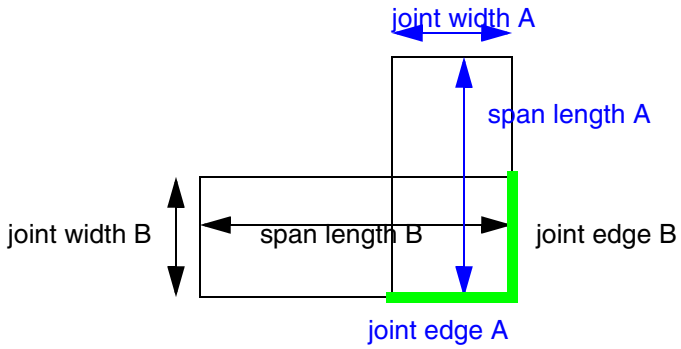
  The first value is the required extension, in user units, of the interconnect or routing layer shape past the cut shape to one of the joint edges. The second value is the required extension to both joint edges. If either requirement is met, the constraint is satisfied.

Parameters

- `cutClass` (`DualValue`, optional) identifies the cut class by width and length. If a cut layer has more than one cut class, this parameter is required, and one constraint should be specified for each individual cut class.

- `jointWidth` (`Value`, required) specifies that the constraint applies only to joint edges with width less than this parameter value, that are not end-of-line edges with length equal to the wire width.
- `spanLength` (`Value`, required) specifies that the constraint applies only to two consecutive joints whose lengths are greater than this parameter value.

Illustration of minViaJointExtension Parameters



Examples

In this example, the minimum extension of `Metal1` over V0 must be either 0.05 μm to one joint edge or 0.03 μm to both joint edges for joints with width less than 0.15 μm and span length greater than 0.1 μm.

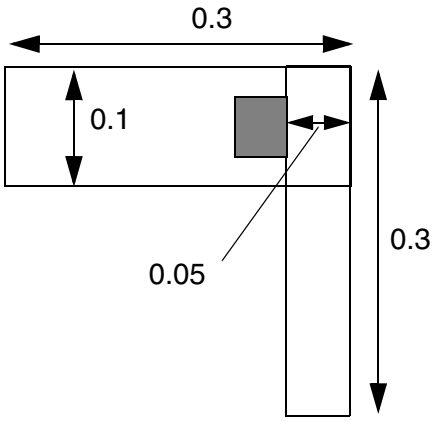
Format	Example

Tcl	`set_constraint_parameter -name jointWidth -Value 0.15`
	`set_constraint_parameter -name spanLength -Value 0.1`
	`set_layerpair_constraint -constraint minViaJointExtension \`
	`  -layer1 Metal1 -layer2 V0 -DualValue {0.05 0.03}`

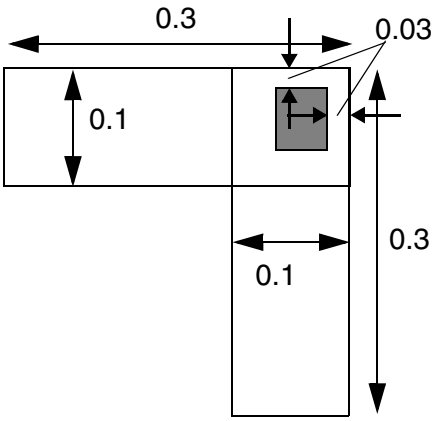
LEF	`PROPERTY LEF58_ENCLOSURETOJOINT`
	`"ENCLOSURETOJOINT 0.05 0.03`
	` JOINTWIDTH 0.15 JOINTLENGTH 0.1 ; " ;`

Virtuoso	`orderedSpacings(`
	`  ( minViaJointExtension Metal1 V0`
	`  'jointWidth 0.15`
	`  'spanLength 0.1`
	`  0.05 0.03)`
	`); orderedSpacings`

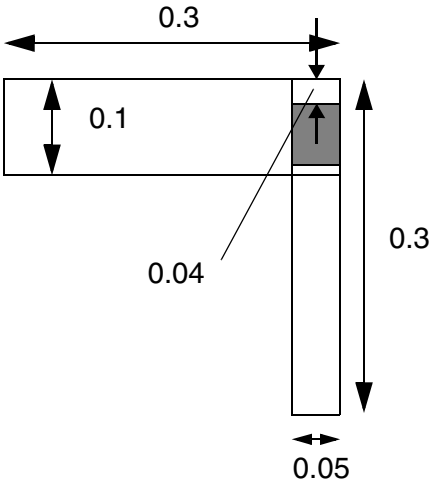
Illustration for minViaJointExtension Example



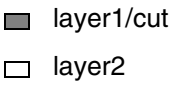
a) OK. Has 0.05 extension to one of the joint edges.



b) OK. Has 0.03 extension to both of the joint edges.



c) Violation. 0.04 and 0.0 extension to the joints does not meet either of the requirements.



The minimum extension of layer2 from the cut to the joint edge must be either 0.05 user units to one joint edge or 0.03 user units to both joint edges.

Related Topics

Extension Constraints

check_extensions

minWireExtension

Specifies the minimum distance a wire on the specified layer must extend past the center of a via to the inside edge of the wire shape. Applies to both cut layers, the layer above and the layer below.

Usually, an OR constraint group should contain only settings of the same constraint with different parameters. If any of the settings is satisfied, the constraint requirement is met. The exception is that minQuadrupleExtension, minDualExtension, and minWireExtension can be bound together in an OR group and if any of the constraints is satisfied, the extension requirement is met.

minWireExtension Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`minWireExtension` constraints have a `Value` that represents the minimum wire extension in user units.

Parameters

- `coincidentAllowed` (`BoolValue`, optional) When true, then shapes can either meet the minimum extension or their edges can be coincident. By default and when false, the edges must meet the specified minimum extension value and *cannot* be coincident.
- `distance` (`Value`, optional) When set, the constraint applies only if there are additional neighboring geometries on layer1 (the extended layer) that are located within the distance specified by this parameter to the extension beyond the via cut.

Examples

This example sets the minimum wire extension on `Metal3` to `1.5` user units.

Format	Example
Tcl	`set_layer_constraint -constraint minWireExtension -layer Metal3 \ -Value 1.5`
Virtuoso	`spacings( (minWireExtension "Metal3" 1.5) )`

Related Topics

Extension Constraints

oaDummyPolyExtension

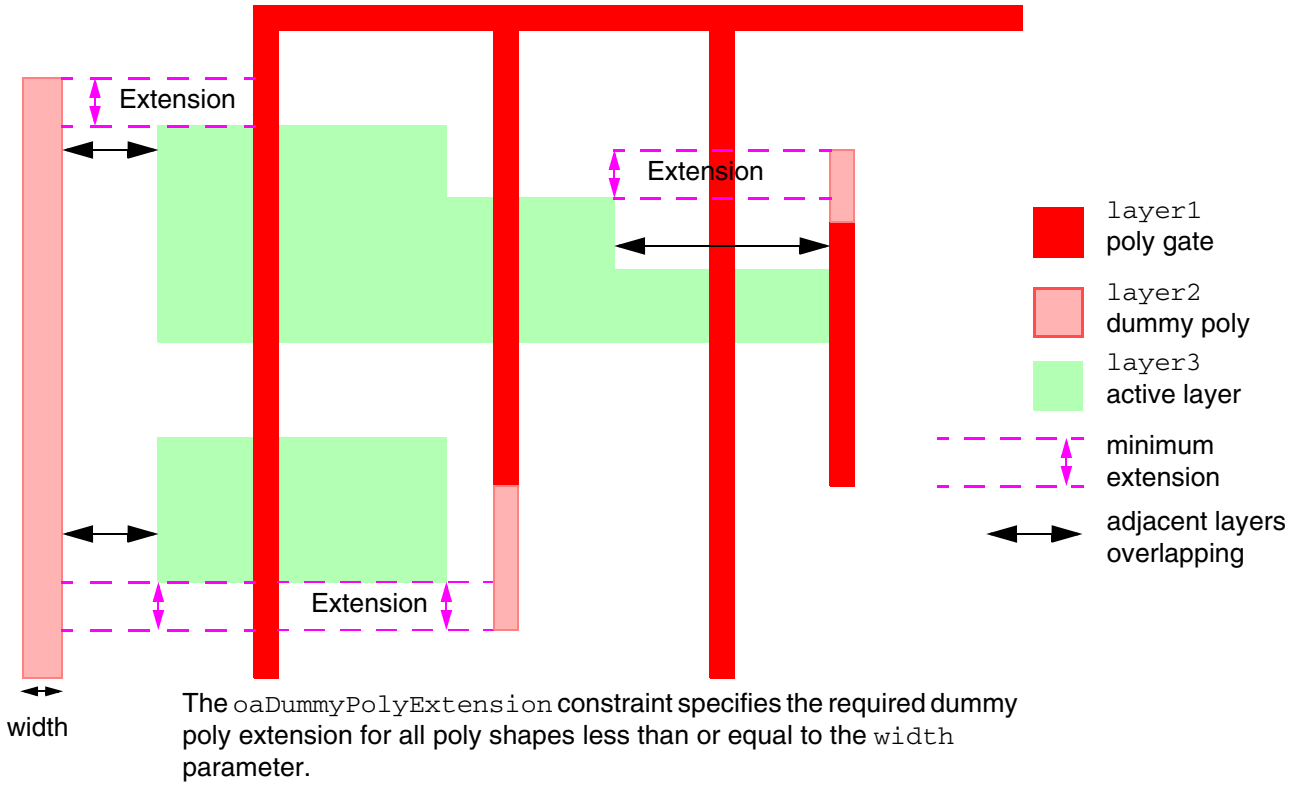
Specifies the required extension of dummy poly shapes beyond the adjacent shapes on the *active* layer. If a gate is already joined to another gate above or below, the constraint does not apply. The first layer is the gate layer, the second is the dummy gate layer, and the third is the active layer. This constraint uses a parameter to specify the maximum ghost poly width at which it applies.

oaDummyPolyExtension Quick Reference

Constraint Type	Layer array
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Extension

Value Type

`oaDummyPolyExtension` constraints have a `Value` that represents the required extension, in user units.



The `oaDummyPolyExtension` constraint specifies the required dummy poly extension for all poly shapes less than or equal to the `width` parameter.

Parameter

The `width` (`Value`) parameter specifies the maximum dummy poly width, in user units, at which the constraint applies.

Examples

This example sets minimum extension of `poly` over `diff` to 1.0 when `polyX` has a width less than 0.5.

Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.5`
	`set_layerarray_constraint -constraint oaDummyPolyExtension \`
	`  -layer_array {"poly1" "polyX" "diff"} -Value 1.0`

Virtuoso	`orderedSpacings(`
	`  (dummyExtension "poly1" "polyX" "diff" 0.5 1.0)`
	`  )`

Related Topics

Extension Constraints

9

Length and Width Constraints

This topic lists the length and width constraints.

allowedBoundaryDimensions
allowedLengthRange
allowedNeighborDiffLayerWidthRange
allowedNeighborOverLayerWidthRange
allowedNeighborWidthRange
allowedWidthRange
discreteWidth
effectiveWidth
maxDiagonalEdgeLength
maxDiffusionLength
maximumLength
maxJogLength
maxWidth
minBumpoutEdgeLength
minConcaveEdgeLength
minConvexEdgeLength
minDiagonalEdgeLength
minDiagonalWidth
minEdgeAdjacentDistance
minEdgeLength
minEdgeMaxCount
minEndOfLineAdjacentToStep
minimumLength
minJogLength
minPerimeter
minProtrusionWidth
minSize
minWidth
oaMinEdgeAdjacentLength
pgFillWidth

Related Topics

Tcl Constraint Commands

Manage Constraints

allowedBoundaryDimensions

Specifies the allowed ranges for PR boundary and virtual hierarchy area boundary dimensions in the specified direction. You can optionally specify that the dimensions must be an exact multiple of a specified `stepSize`. These restrictions are required by certain methodologies to ensure that cells are placeable and flippable.

allowedBoundaryDimensions Quick Reference

Constraint Type	Simple
Value Types	`RangeArrayValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`RangeArrayValue`	Specifies the allowed ranges for the PR boundary dimensions.

Required Parameters

`oaSpacingDirection`	Specifies the measurement direction. The constraint applies only to dimensions measured in this direction. Type: `IntValue` ■ 0 any ■ 1 horizontal ■ 2 vertical

Optional Parameters

`stepSize`

Specifes legal dimensions for a PR boundaries measured in the specifies `oaSpacingDirection`.

The set of legal dimensions is computed as the lower bound (LB) of each range (inclusive) plus a multiple of the `stepSize`, up to the upper bound (UB) of the range if specified. Therefore, the dimension must be equal to `LB + n*stepSize` and <= UB.

Note: If LB is not included in the range specification, the computation is still performed from LB, but the first value calculated (LB itself) is not considered a legal value. The first legal value would be `LB + stepSize`.

If the PR boundary is rectilinear, all dimensions in the given direction must meet the constraint. Specifically, the distance between facing edges perpendicular to spacing direction must meet the constraint. When multiple ranges are specified, `stepSize` applies to all ranges.

If `stepSize` is not specified, the PR boundary needs to fall into one of the specified ranges.

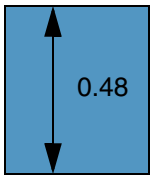
Type: `FltValue`

Examples

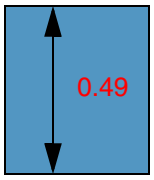
Specifies that the vertical distance between two facing edges must be greater than or equal to `0.24` and less than or equal to `4.8`, and must be `0.24` plus a positive integer multiple of `stepSize`. In other words, it must be `0.24+n*0.048` and within the given range.

```
set_constraint_parameter -name oaSpacingDirection -IntValue 2
set_constraint_parameter -name stepSize -FltValue 0.048
set_constraint -constraint allowedBoundaryDimensions -RangeArrayValue {[0.24 4.8]}
```

```
allowedBoundaryDimensions = [0.24 4.8]
  oaSpacingDirection = 2 (vertical)
  stepSize = 0.048
```

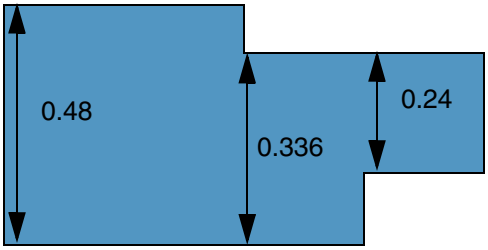


a) PASS. The vertical 0.48 distance between facing edges is within the `allowedBoundaryDimensions` range and is equal to 0.24 plus a positive `stepSize` multiple.
0.48 = 0.24 + 5* 0.048.

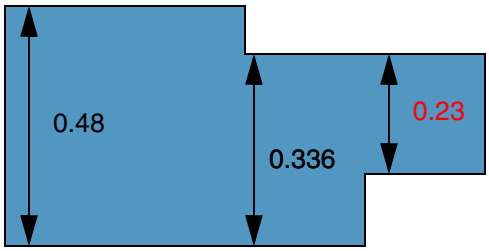


b) FAIL. 0.49 is not a 0.048 `stepSize` multiple from the 0.24 lower bound.

c) FAIL. 0.192 is less than the 0.24 lower bound.



d) PASS. All vertical distances between facing edges are within the `allowedBoundaryDimensions` range and are legal `stepSize` multiples from the 0.24 lower bound (0.24 + *n* *0.048).



e) FAIL. One of the vertical distances is less than the 0.24 lower bound.

Related Topics

Length and Width Constraints

allowedLengthRange

Specifies the allowed length ranges for rectangular shapes on a layer. Non-rectangular shapes are ignored.

allowedLengthRange Quick Reference

Constraint Type	Layer
Value Types	RangeArrayValue, RangeArray1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Types

RangeArrayValue

Specifies the allowed length ranges for the layer.

RangeArray1DTblValue

Specifies a width index and the allowed length ranges for the width on the layer. The widths in this table must be specified in an `allowedWidthRange` constraint.

Examples

Example 1: allowedLengthRange Using RangeArrayValue

Permits three `Metal1` length ranges (0.1, 0.2, and >=0.35).

```
set_layer_constraint -layer Metal1 -constraint allowedLengthRange \
  -RangeArrayValue { "[0.1 0.1]" "[0.2 0.2]" ">= 0.35" }
```

Example 2: allowedLengthRange Using RangeArray1DTblValue

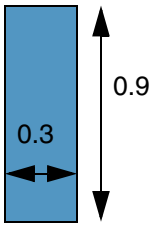
For shapes on layer `Metal1`,

- From `allowedWidthRange` settings, valid widths are 0.3, 0.4, and 0.5.
- From `allowedLengthRange`, find the first row for which the statement is true:

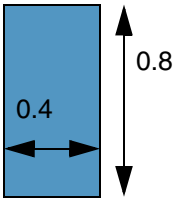
- For width < 0.4, length may be 0.8 or greater than or equal to 1.2.
- For width < 0.5, length may be 0.8, 0.9, or greater than or equal to 1.5 and less than or equal to 2.5.
- For width >= 0.5, length must be greater than or equal to 3.0.

```
set_layer_constraint -constraint allowedWidthRange -layer Metal1 \
  -RangeArrayValue {"[0.3 0.3]" "[0.4 0.4]" "[0.5 0.5]"}
set_layer_constraint -layer Metal1 -constraint allowedLengthRange \
  -row_name width \
  -RangeArray1DTblValue { \
    0.3 2 "[0.8 0.8]" ">= 1.2" \
    0.4 3 "[0.8 0.8]" "[0.9 0.9]" "[1.5 2.5]" \
    0.5 1 ">= 3.0" } \
  -row_interpolation snap_down \
  -row_extrapolation {snap_up snap_down}
```

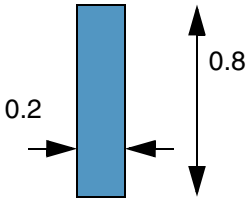
Illustration for allowedLengthRange Using RangeArray1DTblValue



a) FAIL. For width=.3, length must be 0.8 or >= 1.2.



b) PASS. For width=0.4, length must be 0.8, 0.9, or 1.5<=width<=2.5.



c) FAIL. Width=0.2 is not allowed.

Related Topics

Length and Width Constraints

allowedNeighborDiffLayerWidthRange

Specifies the allowed widths for a layer1 shape that depends on the width of another shape on layer2 within a given distance.

allowedNeighborDiffLayerWidthRange Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	RangeArray1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

RangeArray1DTblValue

Specifies a table whose index value is the width in user units of the layer2 shape, and whose table values are the allowed widths, in user units, for the layer1 shape.

For this constraint, index values represent **discrete** values, rather than each entry read as greater or equal to the index value, but less than the next index value.

Required Parameters

`oaDistanceWithin`

Specifies that the constraint applies only if the distance between the shapes is less than this value in user units.

Type: Value

Optional Parameters

`widthRangeArray`

Specifies that the constraint only applies to layer1 shapes with these widths, in user units.

Type: RangeArrayValue

`otherWidthRangeArray`

Specifies that the constraint only applies to layer2 shapes with these widths, in user units.

Type: `RangeArrayValue`

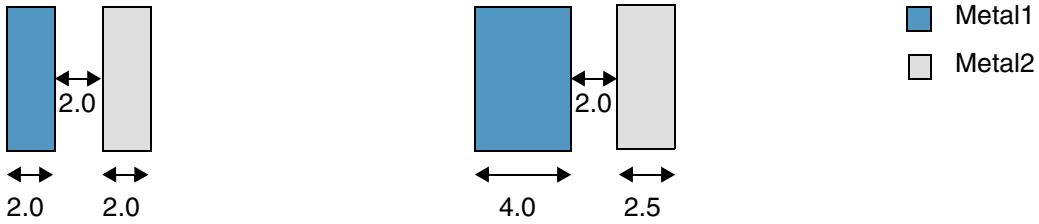
Examples

The following example specifies that

- A `Metal1` shape must be greater than or equal to `2.0` and less than or equal to `3.0` user units in width when it is within 2.5 user units of a `Metal2` shape that is greater than or equal to 2.5 user units in width.
- All `Metal1` shapes must be at least 2.0 user units in width.

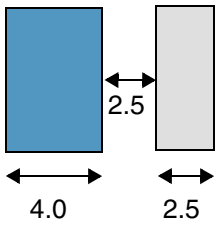
```
set_layer_constraint -constraint minWidth -layer Metal1 -Value 2.0
set_constraint_parameter -name oaDistanceWithin -Value 2.5
set_layer_constraint -layer1 Metal1 -layer2 Metal2 \
  -constraint allowedNeighborDiffLayerWidthRange \
  -RangeArray1DTblValue {2.5 1 [2.0 3.0]}
```

Illustration for allowedNeigborDiffLayerWidthRange



a) PASS. Constraint applies and the width for both shapes is between 2.0 and 3.0 user units.

b) FAIL. The `Metal1` shape width of 4.0 exceeds the allowed width range of 2.0 to 3.0 for a layer2 neighbor of width of 2.5.



c) Constraint does not apply because the `Metal1` and `Metal2` shapes are greater than or equal to 2.5 apart.

Related Topics

Length and Width Constraints

allowedNeighborOverLayerWidthRange

Specifies the allowed widths for a layer1 shape that depends on the width of another shape on layer2, and, optionally, within a given distance, when both shapes overlap the same shape on layer3.

allowedNeighborOverLayerWidthRange Quick Reference

Constraint Type	Layer array
Value Types	RangeArray1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

RangeArray1DTblValue

The index value is the width in user units of the layer2 shape, and whose table values are the allowed widths in user units for the layer1 shape.

Optional Parameters

oaDistanceWithin

Specifies that the constraint only applies if the distance between the shapes is less than this value, in user units.

Type: Value

widthRangeArray

Specifies that the constraint only applies to layer1 shapes with these widths, in user units.

Type: RangeArrayValue

otherWidthRangeArray

Specifies that the constraint only applies to layer2 shapes with these widths, in user units.

Type: RangeArrayValue

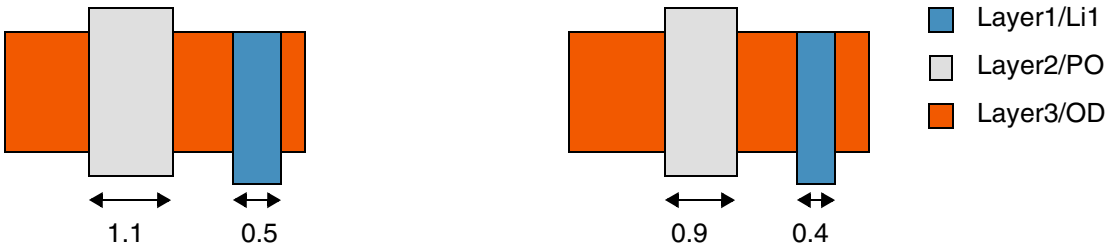
Examples

The following example specifies that

- A layer1 shape must have a width equal to 0.5 or 0.6 user units when it has a neighbor layer2 shape that is at least 1.0 user unit in width and both shapes overlap the same layer3 shape.
- All layer1 shapes must be either 0.4, 0.5, or 0.6 user units in width.

```
set_layer_constraint -constraint allowedWidthRange -layer Li1 \
  -RangeArrayValue {0.4 0.5 0.6}
set_constraint_parameter -name otherWidthRangeArray \
  -RangeArrayValue {">= 1.0"}
set_layer_constraint -layer_array {Li1 PO OD} \
  -constraint allowedNeighborOverLayerWidthRange \
  -RangeArray1DTblValue {1.0 2 0.5 0.6}
```

Illustration for allowedNeighborOverLayerWidthRange



a) PASS. Constraint applies and the layer1 shape has an allowed width (0.5) and a layer2 neighbor of width = 1.0 with both shapes overlapping the same layer3 shape.

b) Constraint does not apply because the layer2 width is < 1.0. The `allowedWidthRange` for layer1 is met (0.4).

Related Topics

Length and Width Constraints

allowedNeighborWidthRange

Specifies the allowed widths for a shape that depends on the width of another shape within a given distance on the same layer.

allowedNeighborWidthRange Quick Reference

Constraint Type	Layer
Value Types	RangeArray1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

RangeArray1DTblValue

The index value is the width of the neighbor shape, in user units, and the table values are the allowed widths, in user units.

Optional Parameters

oaDistanceWithin

Specifies that the constraint only applies if the distance between the shapes is less than this value, in user units.

Type: Value

widthRangeArray

Specifies that the constraint only applies to shapes with these widths, in user units.

Type: RangeArrayValue

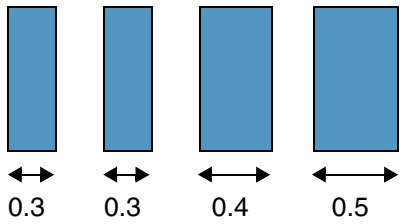
Examples

The following example specifies that

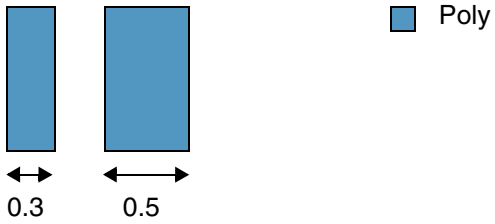
- Poly shapes that are less than 0.4 wide may have neighbor poly shapes with width 0.3 or 0.4.
- Poly shapes that are less than 0.5 wide may have neighbor poly shapes with width 0.3, 0.4, or 0.5.
- Poly shapes that are at least 0.5 wide may have neighbor poly shapes with width 0.4 or 0.5.
- Only poly shapes that are less than 1.5 user units apart and are less than or equal to 1.0 user units in width are considered.

```
set_constraint_parameter -name oaDistanceWithin -Value 1.5
set_constraint_parameter -name widthRangeArray \
  -RangeArrayValue { "<= 1.0"}
set_layer_constraint -constraint allowedNeighborWidthRange \
  -layer Poly -row_name width \
  -RangeArray1DTblValue {0.3 2 0.3 0.4 \
                         0.4 3 0.3 0.4 0.5 \
                         0.5 2 0.4 0.5 }
```

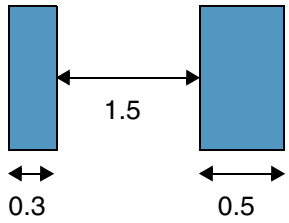
Illustration for allowedNeighborWidthRange



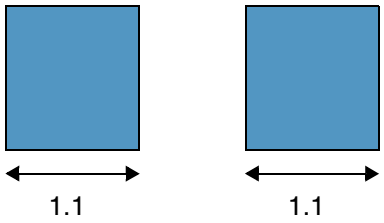
a) PASS. Constraint applies and all shape widths are compatible with their neighbor's width.



b) FAIL. The 0.3 width shape can only have a neighbor shape with a width of 0.3 or 0.4. The 0.5 width shape can only have a neighbor shape of width 0.4 or 0.5.



c) PASS. The constraint does not apply because the distance between the shapes is greater than or equal to 1.5 (`oaDistanceWithin`).



d) Constraint does not apply because both shapes are wider than 1.0.

Related Topics

Length and Width Constraints

allowedWidthRange

Specifies that a particular set of width values are legal for a given layer, instead of defining a minimum and maximum width.

In some advanced node processes, there is an additional requirement for special checking at wire jogs. Specifically, either the vertical or horizontal direction width between two inside corners of a rectilinear object must be greater than or equal to the first width value in the `allowedWidthRange` table *for the corresponding direction* if there are different widths specified for different directions. If there is only one set of allowed widths (or width ranges), then either the vertical or horizontal width between the corners must be greater than or equal to the smallest width value in the width range table.

allowedWidthRange Quick Reference

Constraint Type	Layer
Value Types	`RangeArrayValue`, `OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Types

`RangeArrayValue`	Specifies a list of `RangeValue` values.
`OneDTblValue`	The table headers are index values and the table values are legal widths, in user units, for rectangles on the layer. The last value is interpreted as *greater than or equal to* the given width.

Optional Parameters

`length`	Specifies that the constraint does not apply to wires shorter than this length in user units. Type: `Value`

`oaSpacingDirection`	Specifies the measurement direction. Type: `StringAsIntValue` anyDirection 0 Horizontally and vertically (default) horizontalDirection 1 Horizontally only verticalDirection 2 Vertically only
`orthogonal`	Specifies that either the horizontal or vertical distance between two internal corners of a rectilinear shape must be greater than or equal to the first width value in the `allowedWidthRange` table for the *corresponding* direction if `allowedWidthRange` is specified for both horizontal and vertical directions. If `allowedWidthRange` is only specified for one direction, then either distance must be greater than or equal to the first width value. Type: `BoolValue`
`measurementDirection`	Specifies the direction in which the constraint is measured, regardless of whether this is the width direction or not. The default value, `measureAny`, indicates that the constraint is measured in width direction. Type: `IntValue` `0` measureAny `1` measureHorizontal `2` measureVertical
`interSpace`	Restricts the allowed spacing to integer multiples of this value. Use `withinRange` to limit this parameter to a specified range of values. Type: `IntValue`
`withinRange`	Specifies that the `interSpace` parameter applies only within this range. Type: `RangeValue`

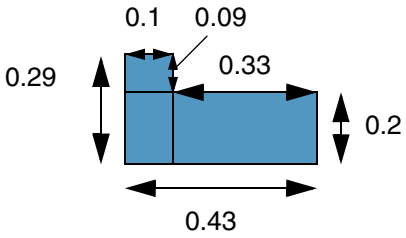
Examples

Example 1: allowedWidthRange Using OneDTblValue

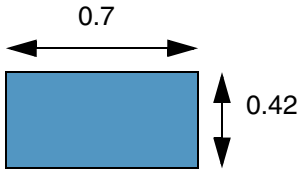
This example sets legal widths for rectangles on `Metal1` to 0.10, 0.20, 0.30 and greater than or equal to `0.40` user units.

```
set_layer_constraint -layer Metal1 -constraint maxWidth -Value 0.4
set_layer_constraint -layer Metal1 -constraint allowedWidthRange \
  -OneDTblValue {0 0.1 1 0.2 2 0.3 3 0.4}
```

Illustration for allowedWidthRange Using OneDTblValue



a) PASS. Polygon is checked as two overlapping rectangles. The constraint is satisfied because the widths for the rectangles, 0.1 and 0.2, are legal widths.



b) PASS. Rectangles with width greater than or equal to 0.4 user units are accepted.

Example 2: allowedWidthRange Using RangeArrayValue

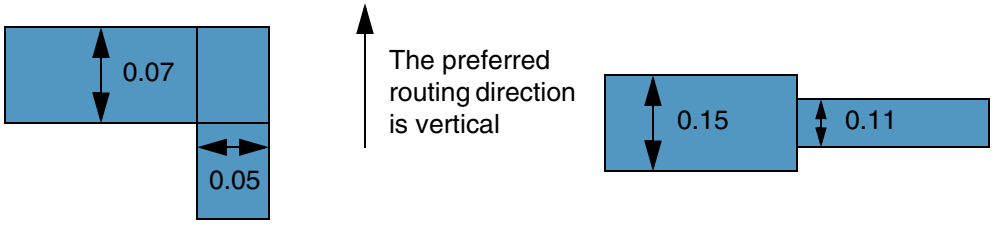
The following example specifies that:

- Either orthogonal distance between two internal corners must be greater than or equal to the smallest width in the corresponding direction.
- Legal widths for vertical wires measured in user units in the horizontal direction are 0.05, 0.07, 0.12, and 0.15 user units.
- Legal widths for horizontal wires measured in the vertical direction are greater than or equal to 0.10 and less than or equal to 0.15 user units.

Note: LEF supports only *exact* widths in WIDTHTABLE, except for the last value, which is interpreted as "greater than or equal to" the value.

```
# preferred direction vertical; oaSpacingDirection=1 (horizontal)

set_constraint_parameter -name oaSpacingDirection 1
set_constraint_parameter -name orthogonal true
set_layer_constraint -layer Metal1 -constraint allowedWidthRange \
  -RangeArrayValue {0.05 0.07 0.10 0.12 0.15}

# wrong way is horizontal, oaSpacingDirection=2 (vertical)

set_constraint_parameter -name oaSpacingDirection 2
set_layer_constraint -layer Metal1 -constraint allowedWidthRange \
  -RangeArrayValue {"[0.10 0.15]"}
```

Illustration for allowedWidthRange Using RangeArrayValue



a) FAIL. 0.07 is not a legal width for a horizontal wire.

b) PASS. 0.15 and 0.11 are legal horizontal wire widths (0.1 <= width <= 0.15).

c) PASS, the right direction width of 0.05 (>= 0.05) is met although the wrong way direction width of 0.03 fails (< 0.10).

d) FAIL, touching wires are subject to the rule, and the right direction width of 0.04 (< 0.05) fails.

Related Topics

Length and Width Constraints

discreteWidth

This constraint is obsolete. Use allowedWidthRange instead.

effectiveWidth

Specifies, in combination with the DESIGNRULEWIDTH parameter, the effective width of a blockage on a specific layer.

effectiveWidth Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`effectiveWidth` constraints have a `Value`.

Related Topics

Length and Width Constraints

maxDiagonalEdgeLength

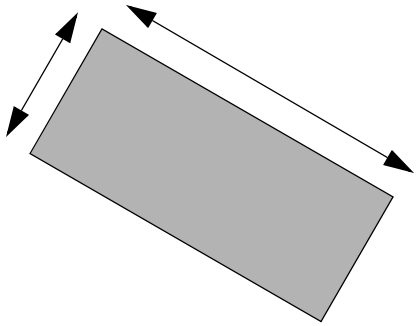
Specifies the maximum length of a diagonal edge on the specified layer.

maxDiagonalEdgeLength Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`maxDiagonalEdgeLength` constraints have a Value that is the maximum length of a diagonal edge on the layer.



The length of any diagonal edge on the specified layer must be less than or equal to the value specified by this constraint.

Examples

This example sets the maximum diagonal edge length on `poly1` to `3.0`.

Format	Example
Tcl	`set_layer_constraint -constraint maxDiagonalEdgeLength \ -layer poly1 -Value 3.0`
LEF	
Virtuoso	`spacings( (maxDiagonalEdgeLength "poly1" 3.0) )`

Related Topics

Length and Width Constraints

maxDiffusionLength

Specifies the maximum diffusion length between two cut shapes and between a cut shape and a line-end of the diffusion shape.

maxDiffusionLength Quick Reference

Constraint Type	Layer-pair (non-symmetric)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width
Group Operators	AND, OR

Value Type

`Value`	Specifies the maximum diffusion length, in user units, between two cut shapes and between a cut shape and a line-end of the diffusion shape.

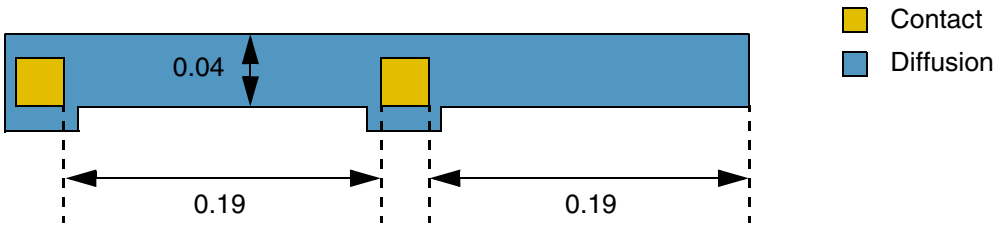
Required Parameters

`'width` *`f_width`*	Specifies that the constraint applies only if the width of the diffusion shape is less than this value.

Examples

Requires that the distance between two Contact edges and a Contact edge and the nearest Diffusion line-end must be less than 0.2 if the width of the Diffusion shape is less than `0.05`.

```
set_constraint_parameter -name width -Value 0.05
set_layerpair_constraint -constraint maxDiffusionLength \
  -layer1 Diffusion -layer2 Contact -Value 0.2
```



PASS. The width of the shape on Diffusion layer is 0.04 (<0.05) and the distance between the two Contact edges and the Contact edge and the Diffusion line-end is 0.19 (<0.2).

Related Topics

Length and Width Constraints

maximumLength

Specifies the maximum length, in user units, for any geometry on the layer.

maximumLength Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`maximumLength` constraints have a `Value` that represents the maximum length for a geometry in user units. The length of a shape is defined as the larger of the shape's two dimensions.

Examples

This example sets the maximum length for shapes on `Metal1` to 5.0 μm.

```
set_layer_constraint -constraint maximumLength -layer Metal1 -Value 5.0
```

Related Topics

Length and Width Constraints

maxJogLength

Specifies the maximum length, in user units, for a wire jog, measured center point to center point. By default, the maximum jog length is infinite.

maximumLength Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`maxJogLength` constraints have a `Value` that represents the maximum length for a wire jog, in user units, measured center point to center point

Examples

This example sets the maximum length for wire jogs on `Metal1` to 15.0 µm.

```
set_layer_constraint -constraint maxJogLength -layer Metal1 -Value 15.0
```

Related Topics

Length and Width Constraints

maxWidth

Sets the maximum width for any shape on the specified layer.

maxWidth Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`maxWidth` constraints have a Value representing the maximum width for any shape on the specified layer.

Examples

This example sets the maximum width of shapes on `Metal3` to 2.5 μm.

Format	Example
Tcl	`set_layer_constraint -constraint maxWidth -layer Metal3 -Value 2.5`

LEF	`LAYER Metal3`
	`  TYPE ROUTING ;`
	`  MAXWIDTH 2.5 ;`

Virtuoso	`spacings(`
	`  (maxWidth "Metal3" 2.5)`
	`)`

Related Topics

Length and Width Constraints

minBumpoutEdgeLength

Specifies the minimum length for an edge that touches two outer vertices and a minimum step edge.

minBumpoutEdgeLength Quick Reference

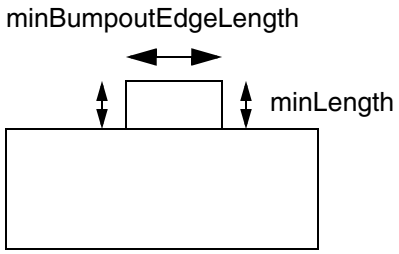
Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`minBumpoutEdgeLength` constraints specify the minimum length, in user units, for an edge that touches two outer vertices and a minimum step edge shorter than `minLength`.

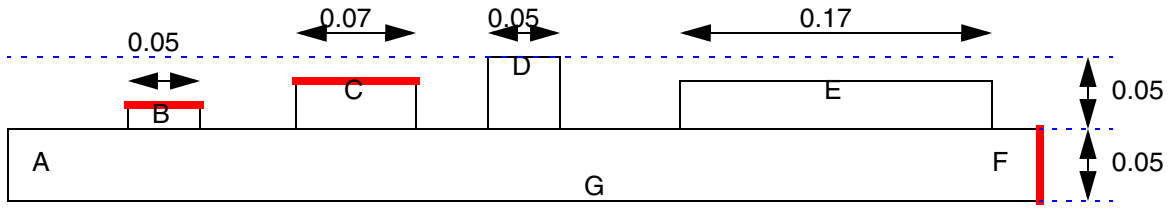
Parameter

- `minLength` (`Value`, optional) specifies that the constraint applies if the bumpout edge touches a minimum step edge that is shorter than this length.



Examples

This example requires an edge to be greater than or equal to 0.15 user units in length if it touches two outside vertices and an edge that is shorter than 0.05 user units.



Edge	Edge Length	Adjacent Edge1	Adjacent Edge2	Status
A	0.05	0.59	0.07	Constraint does not apply
B	0.05	0.01	0.01	Violation, edge length < 0.15
C	0.07	0.03	0.03	Violation, edge length < 0.15
D	0.05	0.05	0.05	Constraint does not apply
E	0.17	0.03	0.03	OK, edge length >= 0.15
F	0.05	0.03	0.59	Violation, edge length < 0.15
G	0.59	0.05	0.05	Constraint does not apply

Format	Example

Tcl	`set_constraint_parameter -name minLength -Value 0.05`
	`set_layer_constraint -constraint minBumpoutEdgeLength \`
	`  -layer Metal3 -Value 0.15`

Related Topics

Length and Width Constraints

minConcaveEdgeLength

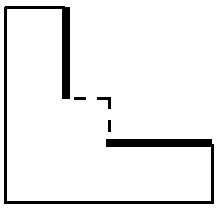
Specifies the minimum edge length for an edge that is in a concave corner of a shape on the layer.

minConcaveEdgeLength Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

The `minConcaveEdgeLength` constraint has a `Value` that is the minimum edge length, in user units, for an edge that is in a concave corner of a shape on the layer.



This shape shows an inside corner because the two edges that are >= `minConcaveEdgeLength` (shown with thick lines) that abut the consecutive edges that are < `minConcaveEdgeLength` (shown with dashed lines) form an inside corner (or concave shape).

Parameter

- `lengthSum` (`Value`, optional) specifies that the constraint does not apply if the sum of consecutive edges that are shorter than the minimum edge length value is less than the value of this parameter.

Examples

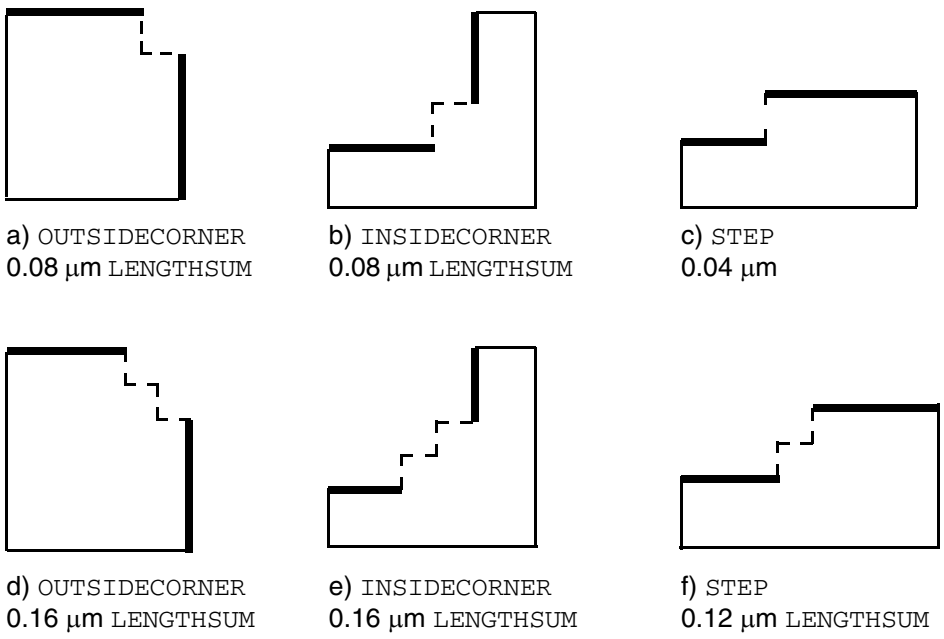
Format	Example
Tcl	`set_constraint -layer Metal2 -constraint minConcaveEdgeLength \ -value 0.05`
LEF	`MINSTEP 0.05 INSIDECORNER ;`
Virtuoso	`spacings( (minInsideCornerEdgeLength "Metal2" 0.05) )`

Sets the minimum inside corner edge length on `Metal2` to 0.05. The results of the specified constraint on shapes in the following figure are:

- Shapes `b` and `e` are violations because their consecutive edges are less than 0.05 μm.
- Shapes `a`, `c`, `d` and `f` are not inside corner checks.

Concave and Convex Edge Examples

Note: All dashed edges are 0.04 μm in length.



a) `OUTSIDECORNER` 0.08 μm `LENGTHSUM`
b) `INSIDECORNER` 0.08 μm `LENGTHSUM`
c) `STEP` 0.04 μm
d) `OUTSIDECORNER` 0.16 μm `LENGTHSUM`
e) `INSIDECORNER` 0.16 μm `LENGTHSUM`
f) `STEP` 0.12 μm `LENGTHSUM`

Related Topics

Length and Width Constraints

minConvexEdgeLength

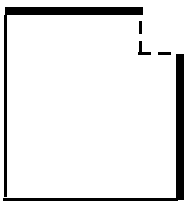
Specifies the minimum edge length for an edge that is in a convex corner of a shape on the layer.

minConvexEdgeLength Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

The `minConvexEdgeLength` constraint has a `Value` that is the minimum edge length, in user units, for an edge that is in a convex corner of a shape on the layer.



This shape shows an outside corner because the two edges that are >= `minConvexEdgeLength` (shown with thick lines) that abut the consecutive edges that are < `minConvexEdgeLength` (shown with dashed lines) form an outside corner (or convex shape).

Parameter

- `lengthSum` (`Value`, optional) specifies that the constraint does not apply if the sum of consecutive edges that are shorter than the minimum edge length value is less than the value of this parameter.

Examples

This example sets the minimum outside corner edge length on `Metal2` to 0.05 and the maximum edge length sum to 0.15. The results of the specified constraint on shapes in Concave and Convex Edge Examples are:

- Shape `a` is legal because its consecutive edges are less than 0.05 μm, and the total length of the edges is less than or equal to 0.15 μm.

- Shape `d` is a violation because even though its consecutive edges are less than 0.05 μm, the total length of the edges is greater than 0.15 μm.
- Shapes `b`, `c`, `e`, and `f` do not have outside corners to check.

Format	Example

Tcl	`set_constraint_parameter -name lengthSum -Value 0.15`
	`set_constraint -layer Metal2 -constraint minConvexEdgeLength \`
	`  -Value 0.05`

LEF	`MINSTEP 0.05 OUTSIDECORNER LENGTHSUM 0.15;`

Virtuoso	`spacings(`
	`  (minOutsideCornerEdgeLength "Metal2" (0.15 0.05))`
	`  )`

Related Topics

Length and Width Constraints

minDiagonalEdgeLength

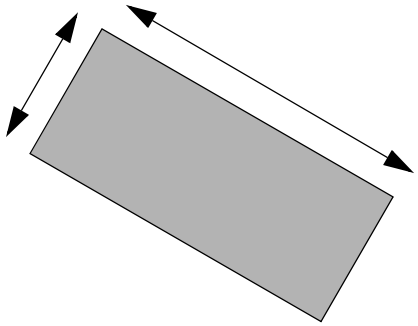
Specifies the minimum length of a diagonal edge on the specified layer.

maxDiagonalEdgeLength Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`minDiagonalEdgeLength` constraints have a Value that is the minimum length of a diagonal edge on the layer.



The length of any diagonal edge on the specified layer must be greater than or equal to the value specified by this constraint.

Examples

This example sets the minimum diagonal edge length on `poly1` to `3.0`.

Format	Example
Tcl	`set_layer_constraint -constraint minDiagonalEdgeLength \` `  -layer poly1 -Value 3.0`
LEF	`LAYER poly1` `  DIAGMINEDGELENGTH 3.0 ;`

Format	Example

Virtuoso	`spacings(`
	`  (minDiagonalEdgeLength "poly1" 3.0)`
	`)`

Related Topics

Length and Width Constraints

minDiagonalWidth

Sets the minimum width of diagonal shapes on a layer.

minDiagonalWidth Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`minDiagonalWidth` constraints have a Value that represents the minimum diagonal width for any geometry.

The width of diagonal shapes on the specified layer must be greater than or equal to the value specified by this constraint.

Examples

This example sets the minimum width of diagonal shapes on `Metal1` to `1.0`.

Format	Example
Tcl	`set_layer_constraint -constraint minDiagonalWidth \` `-layer Metal1 -Value 1.0`
LEF	`LAYER Metal1` `DIAGWIDTH 1.0`

Virtuoso	`spacings(`
	`(minDiagonalWidth "Metal1" 1.0`
	`)`

Related Topics

Length and Width Constraints

minEdgeAdjacentDistance

Specifies the minimum length, in user units, for one of the edges between one set of consecutive minimum step edges and another set of consecutive minimum step edges. This constraint also specifies the minimum step edge length, in user units, and optionally, specifies how many minimum step edges must be connected for this constraint to apply. This constraint is usually applied in combination with the minEdgeMaxCount constraint.

minEdgeAdjacentDistance Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

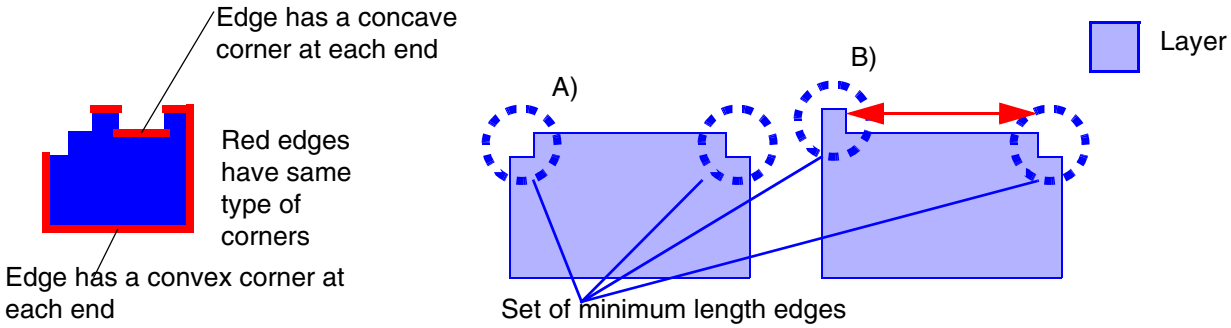
The `minEdgeAdjacentDistance` constraint is a `Value` that specifies the minimum length for one of the edges between one set of consecutive minimum step edges and another set of consecutive minimum step edges.

Parameters

The `minEdgeAdjacentDistance` constraint has the following parameters:

- `maxLength` (`Value`) specifies that edges shorter than this value, in user units, are considered to be minimum step edges.
- `maxEdgeCount` (`IntValue`) specifies the maximum number of consecutive minimum step edges that are allowed. If `count` equals 0, then no edge can be less than `length`. Most tools only allow a value of `0`, `1`, or `2`.
- `exceptSameCorner` (`BoolValue`, optional) specifies that the constraint does not apply for an edge that has the same type (concave or convex) of 90-degree corner at each end.

Illustration of ExceptSameCorner



Minimum edge adjacent distance with except same corners only applies to B) which has different corners at either end.

Examples

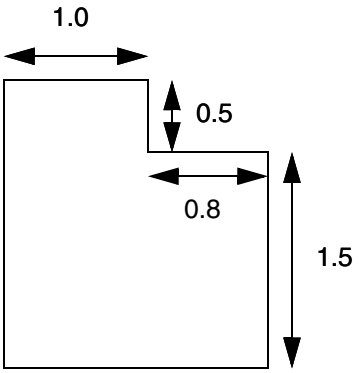
In the following figure, between minimum step edges that are less than `0.6` user units each, there must be an edge that is at least `1.0` user units.

Format	Example

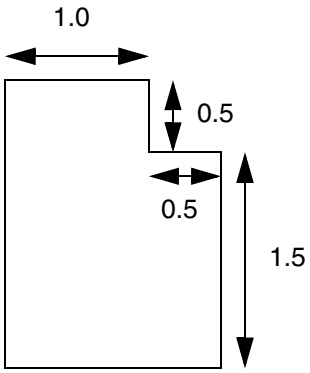
Tcl	`set_constraint_parameter -name maxLength -Value 0.6`
	`set_constraint_parameter -name maxEdgeCount -IntValue 1`
	`set_layer_constraint -layer Metal1 \`
	`  -constraint minEdgeAdjacentDistance -Value 1.0 -hardness hard`

LEF	`PROPERTY LEF58_MINSTEP \` `  "MINSTEP 0.6 MAXEDGES 1 MINBETWEENLENGTH 1.0 ;" ;`

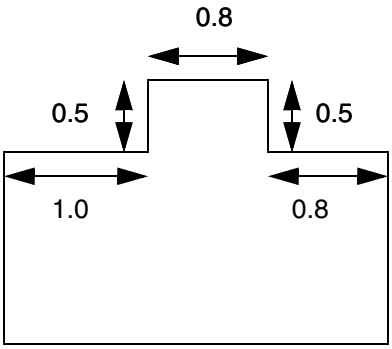
Examples Using the minEdgeAdjacentDistance Constraint



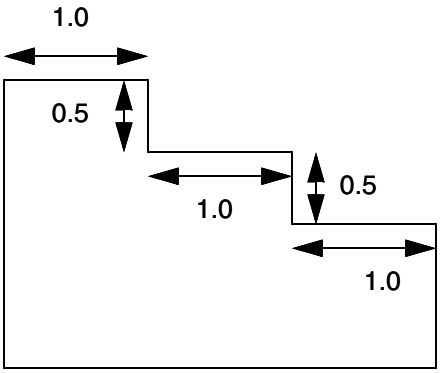
a) OK: An edge < 0.6 μm long has a > 1.0 edge between it and any other < 0.6 edge.



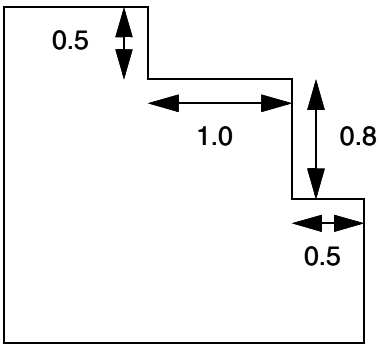
b) Violation: An edge < 0.6μm long has no edge >= 1.0 μm before another < 0.6μm edge.



c) Violation: The < 0.6 μm edges have a < 1.0 μm edge between them.



d) OK: The < 0.6 μm edges have an edge between them that is >= 1.0μm.



e) OK: The < 0.6 μm edges have an edge between them that is >= 1.0μm.

Related Topics

Length and Width Constraints

minEdgeLength

Sets the minimum edge length for a shape on the layer.

minEdgeLength Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width
Group Operators	AND, OR

Value Type

`minEdgeLength` constraints have a `Value` that represents the minimum edge length in user units.

Parameters

The `minEdgeLength` constraint has these optional parameters:

- `lengthSum` (`Value`) specifies that the constraint does not apply if the sum of consecutive edges that are shorter than the minimum edge length value is less than the value of this parameter.
- `cutToMetalSpacing` (`Value`) specifies that the constraint applies only when there is a cut shape (`enclosedCut`) on the layer above or below the metal layer that is less than or equal to this distance from the metal edge.
- `enclosedCut` (`StringAsIntValue`) specifies which cuts are considered with `cutToMetalSpacing` as given in the following table.

enclosedCut Values

String	Integer Value	Constraint applies when:
`cutIsAboveAndBelow`	0	Cut shape on layer above or below

String	Integer Value	Constraint applies when:
`cutIsBelow`	1	Cut shape on layer below
`cutIsAbove`	2	Cut shape on layer above

Examples

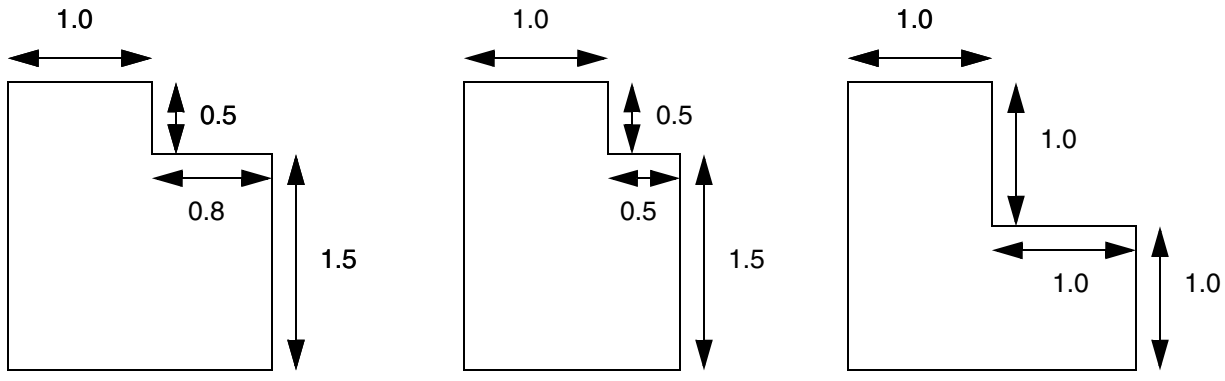
Example1—minEdgeLength without lengthSum

This example sets the minimum edge length for shapes on `Metal1` to 1.0.

Format	Example
Tcl	`set_layer_constraint -layer Metal1 \` `  -constraint minEdgeLength -Value 1.0 -hardness hard`
LEF	`MINSTEP 1.0 STEP ;`

The following figure illustrates how the constraint setting affects three shapes.

Illustration of minEdgeLength without lengthSum



a) Violation. Two edges are less than 1.0 in length.

b) Violation: Two edges are less than 1.0 in length.

c) No violation. All edges are greater than or equal to the `minEdgeLength`.

Example2—minEdgeLength with lengthSum

This example sets the minimum edge length for shapes on `Metal1` to 1.0 and excludes consecutive minimum steps whose total length is less than 1.1.

Format	Example

Tcl	`set_constraint_parameter -name lengthSum 1.1`
	`set_layer_constraint -layer Metal1 \`
	`  -constraint minEdgeLength -Value 1.0 -hardness hard`

LEF	`LAYER Metal1` `  MINSTEP 1.0 STEP LENGTHSUM 1.1 ;`

Using these settings for `minEdgeLength` and `lengthSum`, shapes b and c in the following figure are legal. For shape b, the consecutive edges that are shorter than `minEdgeLength` have a total length of 1.0, which is less than the `lengthSum` value, so the constraint is not applied. Shape a is a violation because the total length of the consecutive minimum length edges is greater than the `lengthSum` value.

Illustration of minEdgeLength with Cut Shape Dependency

This example sets the minimum edge length for shapes on `Metal2` to 0.025 whenever there is a cut shape on the layer above the metal that is less than or equal to 0.04 from a `Metal2` shape.

Format	Example

Tcl	`set_constraint_parameter -name cutToMetalSpacing -Value 0.04`
	`set_constraint_parameter -name enclosedCut \`
	`  -StringAsIntValue cutIsAbove\`
	`set_layer_constraint -layer Metal2 -constraint minEdgeLength \`
	`  -Value 0.025`

Related Topics

Length and Width Constraints

minEdgeMaxCount

Specifies the maximum allowed number of consecutive small edges that is permitted in a step to satisfy the `minEdgeLength` constraint.

minEdgeMaxCount Quick Reference

Constraint Type	Layer
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`minEdgeMaxCount` constraints have an `IntValue`.

Parameter

- `maxLength` (`Value`) specifies the maximum length, in user units, for each small edge in the step.

Examples

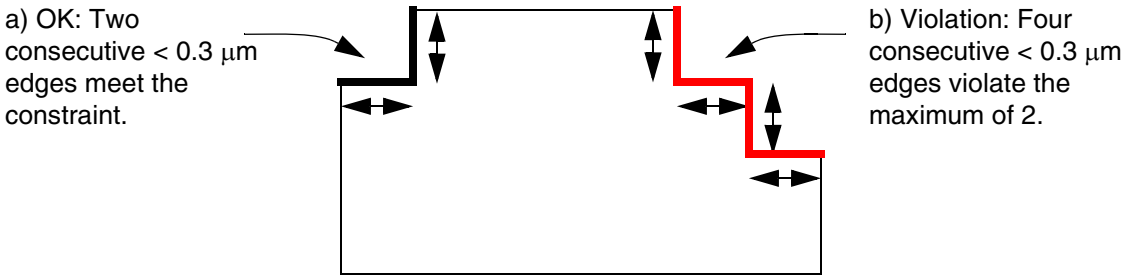
The example in the following figure permits up to two consecutive edges of less than `0.3`.

Format	Example

Tcl	`set_constraint_parameter -name maxLength -Value 0.3`
	`set_layer_constraint -layer Metal1 -constraint minEdgeMaxCount \`
	`  -IntValue 2 -hardness hard`

LEF	`MINSTEP 0.3 MAXEDGES 2 ;`

Example for minEdgeMaxCount Constraint



Related Topics

Length and Width Constraints

minEndOfLineAdjacentToStep

A minimum step rule optionally requires a certain minimum length for end-of-line edges that are adjacent to a minimum edge. If a minimum step edge has an adjacent edge that is also an end-of-line edge, that end-of-line edge must not be shorter than a certain value.

minEndOfLineAdjacentToStep

Constraint Type	Layer
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

IntValue	Specifies in user units the required minimum length of an end-line edge adjacent to an edge less than the `maxLength` value.

Required Parameters

`maxLength`	Specifies the maximum length of the adjacent edge; the constraint applies if the end-of-line edge has an adjacent edge of length less than this value. Type: IntValue

Optional Parameters

`exceptAdjacentLength`	Specified that the no adjacent EOL minimum step rule does not apply if the other neighbor edge of the minimum step edge has length greater than or equal to `adjacentLength`. Type: IntValue

`noBetweenEol`	Specifies that any end-of-line with length less than the constraint value cannot have adjacent steps of length less than the `maxLength` parameter. Type: `BoolValue` (default is false)
`concaveCorner`	Specifies that both of the neighbor edges of a EOL have concave corner at the other end. Type: `BoolValue` (default is false)
`adjacentLength`	Specifies that the constraint applies only if the other neighbor edge of the minimum step edge has length greater than or equal to this value. Type: `IntValue`

Related Topics

Length and Width Constraints

minimumLength

Specifies the minimum orthogonal length for any geometry on a layer.

minimumLength Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`minimumLength` constraints have a `Value` that represents the minimum orthogonal length for any shape on the layer.

The length of the geometry must be greater than or equal to the `minimumLength` constraint.

Examples

This example sets the minimum orthogonal length for shapes on `Metal2` to `0.6` user units.

```
set_layer_constraint -constraint minimumLength -layer Metal2 -Value 0.6
```

Related Topics

Length and Width Constraints

minJogLength

Specifies the minimum length, in user units, for a wire jog, measured center point to center point. If `minWidth` is not set, the default minimum jog length is 0; otherwise, the default is `minWidth*2`.

maximumLength Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`minJogLength` constraints have a `Value` that represents the minimum length for a wire jog, in user units, measured center point to center point.

Examples

This example sets the minimum length for wire jogs on `Metal1` to 5.0 μm.

```
set_layer_constraint -constraint minJogLength -layer Metal1 -Value 15.0
```

Related Topics

Length and Width Constraints

minPerimeter

Specifies the minimum perimeter of a shape on a given layer.

minPerimeter Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

The `minPerimeter` constraint has a Value in user units.

Example of minPerimeter

Examples

The following example sets a minimum perimeter of 1.5 for `Metal1` shapes.

```
set_layer_constraint -constraint minPerimeter -Value 1.5
```

Related Topics

Length and Width Constraints

minProtrusionWidth

Specifies the minimum width for a protrusion that connects to a wire based on the length of the protrusion and the width of the wire it connects to. The following must be true for the constraint to apply:

- The protrusion must connect to a wire.
- The length of the protrusion must be less than the *protrusionLength* in the constraint specification.
- The width of the wire the protrusion connects to must be less than or equal to the *wireWidth* in the constraint specification.

minProtrusionWidth Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

minProtrusionWidth constraints have a Value that represents the minimum width, in user units, for the protrusion to apply when the criteria is met.

Format	Syntax

Tcl	`set_constraint_parameter -name width -Value f_wireWidth`
	`set_constraint_parameter -name length -Value f_protrusionLength`
	`set_layer_constraint -constraint minProtrusionWidth \`
	`-layer s_layer -Value f_protrusionWidth`

LEF	`LAYER s_layer`
	`TYPE ROUTING ;`
	`PROTRUSIONWIDTH width1 LENGTH length WIDTH width2 ;`

Virtuoso	`spacings(`
	`( protrusionWidth tx_layer`
	`( g_protrusionLength g_wireWidth g_protrusionWidth ) )`
	`)`

Parameters

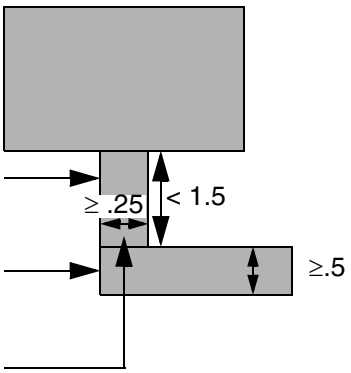
- `width` (`Value`) is the minimal width, in user units, of the wide wire for the constraint to apply.
- `length` (`Value`) is the length, in user units, that the protrusion length must be less than for the constraint to apply.

Examples

This example sets the minimum protrusion width to `0.25` on `Metal3` protrusions on that meet the following requirements:

- Have a length shorter than `1.5` user units.
- Connect to a wire with a width greater than or equal to `0.5`.

Assume that `length` is set to `1.5`, `width` is set to `0.5`, and `minProtrusionWidth` is set to `0.25` on this layer



Protrusion with length less than `length` argument and protrusion connects to wire with width greater than or equal to `width` argument, then the width of the protrusion must be greater than or equal to the `minProtrusionWidth` constraint

The width of a protrusion connecting to a wire must be greater than or equal to the value of this constraint when both of the following conditions are met:
1. The protrusion length is less than `length`
and
2. The wire width is greater than or equal to `width`

Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.5`
	`set_constraint_parameter -name length -Value 1.5`
	`set_layer_constraint -constraint minProtrusionWidth \`
	`-layer Metal3 -Value 0.25`

Format	Example

LEF	`LAYER Metal3`
	` TYPE ROUTING ;`
	` PROTRUSIONWIDTH 0.5 LENGTH 1.5 WIDTH 0.25 ;`

Virtuoso	`spacings(`
	`  (protrusionWidth "Metal3" (1.5 0.5 0.25)`
	`)`

Related Topics

Length and Width Constraints

minSize

Specifies the minimum width and length of a rectangle that must be able to fit inside any polygon on the layer.

In some processes, the minimum allowed size for a shape is different for rectangular shapes as compared to non-rectangular shapes. The constraint can optionally apply only to rectangular shapes.

minSize Quick Reference

Constraint Type	Layer
Value Types	DualValueTbl
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

DualValueTbl	Specifies one or more width/length pairs that represent the dimensions, in user units, of rectangles. This constraint is satisfied if any of the rectangles can fit within the polygon.

Optional Parameter

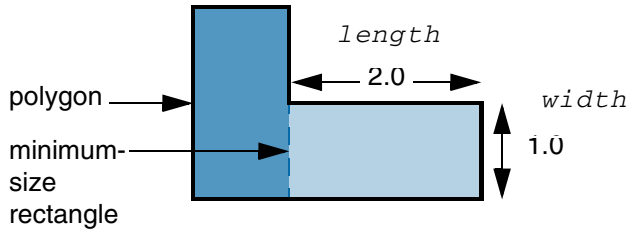
rectOnly	Specifies that the constraint applies only to rectangular shapes. Type: BoolValue

Examples

Example 1: minSize

Requires that a 1.0 by 2.0 rectangle must fit within each Metal1 polygon.

```
set_layer_constraint -constraint minSize -layer Metal1 \
 -DualValueTbl { 1.0 2.0 }
```

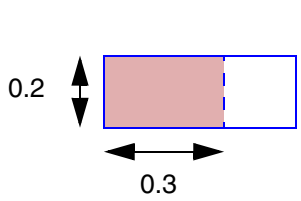


Any polygon on the specified layer must be able to contain a rectangle of the minimum length and width specified by this constraint.

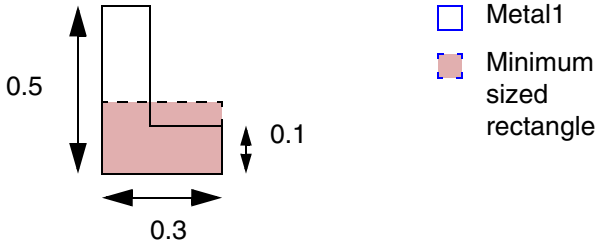
Example 2: minSize with rectOnly parameter

Specifies that a `0.2` by `0.3` rectangle must fit within each `Metal1` rectangle.

```
set_constraint_parameter -name rectOnly -BoolValue true
set_layer_constraint -constraint minSize -layer Metal1 \
 -DualValueTbl { 0.2 0.3 }
```



a) OK. Constraint satisfied. The minimum width rectangle fits inside the layer1 rectangle.



b) OK. Minimum sized rectangle does not fit in layer1 shape but the layer1 shape is a polygon, not a rectangle, so the constraint does not apply.

Related Topics

Length and Width Constraints

minWidth

Determines the minimum orthogonal width, in user units, for shapes on a specified layer.

For some processes, the minimum width constraint does not apply for one or more of these cases:

- Where two shapes touch at a point
- Where two shapes abut for a sufficient overlap.

minWidth Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, net group, design, foundry
Category	Length and Width
Group Operators	AND, OR

Value Type

`minWidth` constraints have a `Value` that represents the minimum width, in user units, for any shape on the layer.

The width of a shape on the layer must be greater than or equal to the value of the `minWidth` constraint.

Parameters

- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1). For information on this parameter, see "Euclidean and Manhattan Spacing Constraints" on page 52.
- `exceptLineTouch` (`DualValueTbl`, optional) specifies that `minWidth` violations caused by two shapes abutting for an overlap distance within the given ranges must be ignored. Each DualValue pair represents a range. Normally, when the overlap distance is less than `minWidth`, a violation occurs. However, if the overlap distance is greater

than or equal to the first value of a DualValue pair and less than or equal to the second value of a DualValue pair, the violation is ignored.

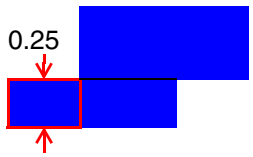
- `exceptPointTouch` (`BoolValue`, optional) specifies that `minWidth` violations caused by two shapes touching at corner points must be ignored.
- `oaSpacingDirection` (`StringAsIntValue`, optional) specifies the measurement direction.

anyDirection	0	Horizontally and vertically (default)
horizontalDirection	1	Horizontally only
verticalDirection	2	Vertically only

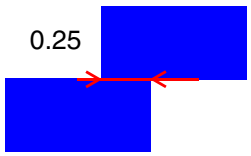
Examples

Example 1—minWidth Fixed Value

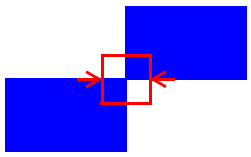
This example sets the minimum width for shapes on `Metal1` to 0.3 µm.



a) Violation. 0.25 shape width < 0.3 `minWidth`.



b) Violation. Two abutting shapes with 0.25 overlap < 0.30 `minWidth`.



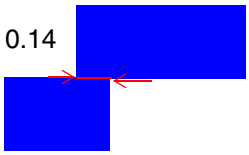
c) Violation due to point touch between the two shapes.

Format	Example
Tcl	`set_layer_constraint -constraint minWidth -layer Metal1 -Value 0.3`
LEF	`LAYER Metal1` `  WIDTH 0.3 ;`

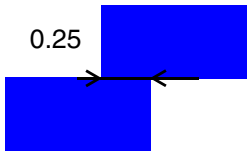
Virtuoso	`spacings(`
	`  (minWidth "Metal1" 0.3)`
	`  )`

Example 2—minWidth with exceptPointTouch and exceptLineTouch

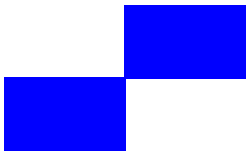
This example sets the minimum width for shapes on `Metal1` to 0.3 μm and excludes violations caused by two shapes touching at a point and by two shapes that abut with an overlap of 0.1, or an overlap >= 0.2 and <= 0.3.



a) Violation. 0.14 overlap between abutting shapes < 0.3 `minWidth` and is not included in `exceptLineTouch` ranges.



b) OK. Two abutting shapes with 0.25 overlap < 0.30 `minWidth` but is included in the second `exceptLineTouch` range.



c) OK because `exceptPointTouch` is set to `true`.

Format	Example

Tcl	`set_constraint_parameter -name exceptPointTouch -BoolValue true`
	`set_constraint_parameter -name exceptLineTouch \`
	`  -DualValueTbl {0.1 0.1 0.2 0.3}`
	`set_layer_constraint -constraint minWidth -layer Metal1 -Value 0.3`

Virtuoso	`spacings(`
	`  (minWidth "Metal1" \`
	`  'exceptPointTouch 'exceptLineTouch (0.1 0.1 0.2 0.3) 0.3 )`
	`  ) ;spacings`

Related Topics

Length and Width Constraints

oaMinEdgeAdjacentLength

Sets the minimum length for edges on a polygon that are adjacent to an edge that is shorter than a specified length. Some processes require specific length values for both adjacent edges. For other processes, the constraint applies if a convex corner lies between two concave corners and if the length of one of the edges which form the convex corner is shorter than a specified length, then the length of the other convex corner edge must be greater than or equal to the value of the constraint.

oaMinEdgeAdjacentLength Quick Reference

Constraint Type	Layer
Value Types	Value, DualValue
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width
Group Operators	AND, OR

Value Types

`oaMinEdgeAdjacentLength` constraints have the following value types:

- Value

  Specifies the minimum length, in user units, of edges adjacent to the short edge when `convexCorner` is `false` or not given. When `convexCorner` is `true`, this specifies the minimum length, in user units, of the longer convex corner edge.

Format	Example

Tcl	`set_constraint_parameter -name maxLength -Value f_maxLength`
	`set_layer_constraint -constraint oaMinEdgeAdjacentLength \`
	`-layer s_layer \ -Value f_length`

LEF	`LAYER s_layer`
	`PROPERTY LEF58_MINSTEP`
	`"MINSTEP f_maxLength MINADJACENTLENGTH f_length ;" ;`

Virtuoso	`spacings(`
	`( minEdgeAdjacentLength s_layer 'maxLength f_maxLength f_length )`
	`)`

- DualValue

  Specifies the minimum lengths, in user units, for the two edges adjacent to the short edge.

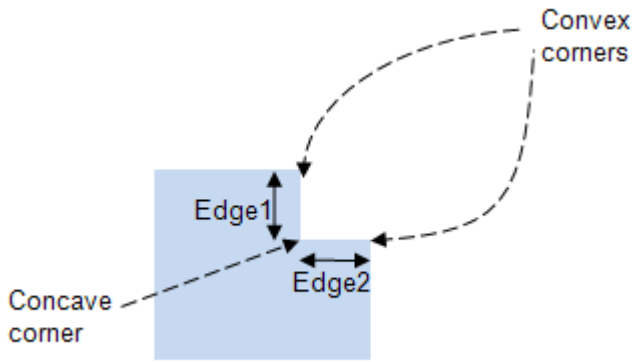
Format	Example

Tcl	`set_constraint_parameter -name maxLength -Value f_maxLength`
	`set_layer_constraint -constraint oaMinEdgeAdjacentLength \`
	`-layer s_layer \ -DualValue {f_length1 f_length2}`

Virtuoso	`spacings(`
	`( minEdgeAdjacentLength s_layer 'maxLength f_maxLength`
	`( f_length1 f_length2 ) )`
	`)`

Parameters

- `convexCorner` (BoolValue, optional) specifies whether the constraint applies when a convex corner lies between two concave corners. When `true`, if the length of one of the edges which form the convex corner is shorter than `maxLength`, then the length of the other convex corner edge must be greater than or equal to the value of the constraint. This parameter applies only when the constraint value type is Value.
- `maxLength` (Value, required) specifies that the constraint applies when the short edge is shorter than this value, in user units.
- `concaveCorner` (BoolValue, optional) specifies that if a concave corner is between two convex corners, and if the length of one of the edges that form the concave corner is less than `maxLength`, then the other length must be greater than or equal to the value of this constraint. The `concaveCorner` parameter is mutually exclusive with the `convexCorner` parameter.



Examples

Example 1—oaMinEdgeAdjacentLength with adjacent length minimums

- On `Metal1`, the length of edges adjacent to an edge shorter than 1.5 must be greater than or equal to 3.0.

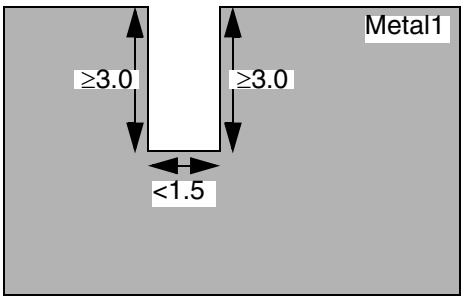
- On `Metal2`, the length of one edge adjacent to an edge shorter than 1.5 must be greater than or equal to 2.0 and the length of a second edge adjacent to the edge shorter than 1.5 must be greater than or equal to 3.0.

Format	Example

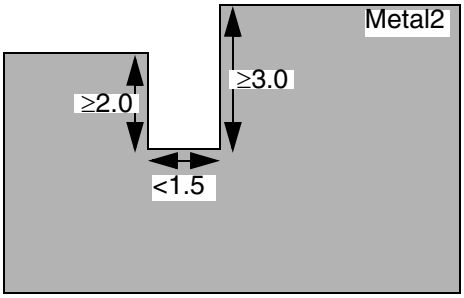
Tcl	`set_constraint_parameter -name maxLength -Value 1.5`
	`set_layer_constraint -constraint oaMinEdgeAdjacentLength \`
	`-layer Metal1 -Value 3.0`
	
	`set_constraint_parameter -name maxLength -Value 1.5`
	`set_layer_constraint -constraint oaMinEdgeAdjacentLength \`
	`-layer Metal2 -DualValue {2.0 3.0}`

LEF	`LAYER Metal1`
	`PROPERTY LEF58_MINSTEP`
	`"MINSTEP 1.5 MINADJACENTLENGTH 3.0 ;" ;`

Virtuoso	`spacings(`
	`( minEdgeAdjacentLength Metal1 'maxLength 1.5 3.0)`
	`( minEdgeAdjacentLength Metal2 'maxLength 1.5 (2.0 3.0) )`
	`) ; spacings`



On Metal1, the length of edges adjacent to an edge shorter than 1.5 must be greater than or equal to 3.0.



On Metal2, the length of one edge adjacent to an edge shorter than 1.5 must be greater than or equal to 2.0 and the length of a second edge adjacent to the edge shorter than 1.5 must be greater than or equal to 3.0.

Example 2—oaMinEdgeAdjacentLength with convexCorner

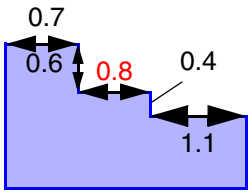
The following example specifies that if a convex corner is between two concave corners, and one edge of the convex corner is less than 0.6 user units, then the other convex corner edge must be longer than or equal to 1.0 user units.

Format	Example

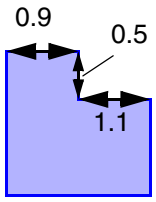
Tcl	`set_constraint_parameter -name maxLength -Value 0.6`
	`set_constraint_parameter -name convexCorner -BoolValue true`
	`set_layer_constraint -constraint oaMinEdgeAdjacentLength \`
	`-layer Metal1 -Value 1.0`

LEF	`LAYER Metal1`
	`PROPERTY LEF58_MINSTEP`
	`"MINSTEP 0.6 MINADJACENTLENGTH 1.0 CONVEXCORNER ;" ;`

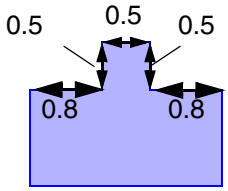
Virtuoso	`spacings(`
	`( minEdgeAdjacentLength Metal1 'maxLength 0.6 'edgeCount 1`
	`'convexCorner 1.0 )`
	`) ;spacings`



a) Violation. Convex corner (0.8&0.4) between 2 concave corners (0.6 & 1.1). Short edge (0.4) of convex corner triggers the constraint but long edge (0.8) of convex corner does not meet 1.0 minimum.



b) OK. The convex corner (0.9&0.5) does not lie between two concave corners.



c) OK. There is no convex corner between two concave corners.

Related Topics

Length and Width Constraints

pgFillWidth

Specifies the width to use when creating power and ground fill stripes.

pgFillWidth Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Length and Width

Value Type

`pgFillWidth` constraints have a `Value` that is the width, in user units, for power and ground fill stripes. If not specified, two times the `minWidth` is used.

Examples

This example sets the width for power and ground fill stripes on `Metal2` to 1.5.

```
set_layer_constraint -constraint pgFillWidth -layer Metal2 -Value 1.5
```

Related Topics

Length and Width Constraints

10

Miscellaneous Constraints

This topic lists the miscellaneous constraints:

allowedWireTypes	diagonalShapesAllowed	edgeMustCoincide
errorLayer	minOneDArrayStructure	oaGateOrientation
rectangularShapeDirection	trimShape	useExistingShapesAsShield

Related Topics

Tcl Constraint Commands

Manage Constraints

allowedWireTypes

Specifies valid wire types for a layer. As a result, only the specified wire types can be set for tracks on that layer. The default value is an empty string, which indicates that a wire type is not set for the tracks on the specified layer (all wire types are allowed).

allowedWireTypes Quick Reference

Constraint Type	Layer
Value Types	StringArrayValue
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

StringArrayValue	Specifies that the string array should contain a list of wire types.

Examples

Sets wire types `1X`, `2X`, and `3X` for tracks on layer `Metal3`.

```
set_layer_constraint -constraint allowedWireTypes \
 -layer Metal3 -StringArrayValue "1X 2X 3X"
```

Related Topics

Miscellaneous Constraints

diagonalShapesAllowed

Specifies whether it is permissible to create diagonal shapes on a specific layer. In general, this constraint is used to indicate whether diagonal shapes are allowed.

diagonalShapesAllowed Quick Reference

Constraint Type	Layer
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

BoolValue	Indicates whether it is an error to create diagonal geometries on the layer. If set to `true`, then diagonal shapes are allowed on the layer.

Examples

Enables diagonal shapes on `Metal1`.

```
set_layer_constraint -constraint diagonalShapesAllowed \
 -layer Metal1 -BoolValue true
```

Related Topics

Miscellaneous Constraints

edgeMustCoincide

Specifies conditions for which edges of one layer must coincide with edges of another layer.

edgeMustCoincide Quick Reference

Constraint Type	Layer pair
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

BoolValue	When set to `true`, an edge on layer1 must coincide with an edge on layer2.

Optional Parameters

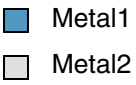
`edgeDirection`	Specifies that the constraint applies only to edges in this direction. Type: IntValue 0 any 1 horizontal 2 vertical
`endOfLineOnly`	Specifies that the constraint applies only if the layer1 edge is between two convex corners and its length is within the specified `edgeLengthRangeArray`. Type: BoolValue
`edgeLengthRangeArray`	Specifies that the constraint applies only if the layer1 edge length is within this range (in user units). Type: RangeArrayValue

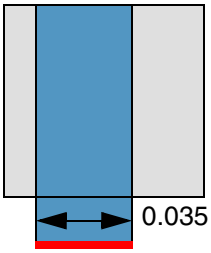
Examples

Specifies that horizontal `Metal1` edges between two convex corners that are less than 0.02 user units in length, or greater than 0.03 user units and less than 0.04 user units in length, must coincide with a `Metal2` edge.

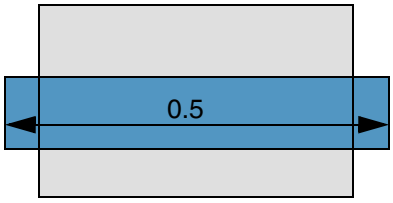
```
set_constraint_parameter -name edgeDirection -IntValue 1
set_constraint_parameter -name endOfLineOnly -BoolValue true
set_constraint_parameter -name edgeLengthRangeArray \
  -RangeArrayValue {"<0.02" "(0.03 0.04)"}
set_layerpair_constraint -layer1 Metal1 -layer2 Metal2 \
  -constraint edgeMustCoincide -BoolValue true
```

```
edgeMustCoincide       true
  edgeDirection        1 (horizontal)
  endOfLineOnly        true
  edgeLengthRangeArray {"<0.02" "(0.03 0.04)"}
```

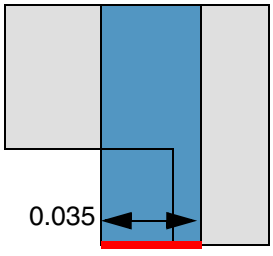




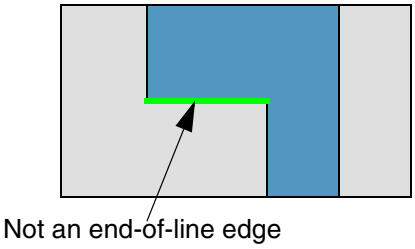
a) FAIL. There are two `Metal1` horizontal edges with length 0.035, within the range given by `edgeLengthRangeArray`, so the constraint applies but is not met because the bottom edge does not coincide with a `Metal2` edge.



b) Constraint does not apply. The two `Metal1` horizontal edges are 0.5 in length and are not in a range given by `edgeLengthRangeArray`.



c) FAIL. There are two `Metal1` horizontal edges with length 0.035, within the range given by `edgeLengthRangeArray`, so the constraint applies, but is not met because the bottom edge is not fully covered by the `Metal2`. edge.



d) PASS. The constraint applies only to end-of-line edges, so the edge shown in green is not considered for this check. The other two horizontal `Metal1` edges coincide with a `Metal2` edge.

Related Topics

Miscellaneous Constraints

errorLayer

Specifies whether it is an error for geometries to exist on a specific derived layer. In general, this constraint is used to indicate derived layers that are an error. An application must not create geometries on an error layer, but should detect when an error layer exists and indicate the error.

errorLayer Quick Reference

Constraint Type	Layer
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

BoolValue	When set to `true`, it is an error for geometries to exist on the layer.

Related Topics

Miscellaneous Constraints

minOneDArrayStructure

Specifies the spacing requirements for perfectly-aligned cuts in a 1D array in the preferred routing direction with a minimum number of cuts. The spacing applies to spacing between cuts on the same layer and on different cut layers. In addition, enclosure rules for the 1D array may differ from single or double cut vias on the same layer and can depend on the cut class.

The first layer (layer1) is the cut layer, the second layer (layer2) is the routing layer above for which extension values will be defined, and the third layer (layer3) is a cut layer for which inter-layer cut array spacing is defined.

minOneDArrayStructure Quick Reference

Constraint Type	Layer array
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

`Value`	Specifies the required exact edge-to-edge spacing between cut shapes in a 1D array.

Required Parameters

`cutClass`	Identifies the cut class by width and length (specified in user units). Type: `DualValue`
`numCuts`	Specifies the minimum number of perfectly-aligned cuts in a row in the preferred routing direction for this structure to qualify as a 1D array. Type: `IntValue`

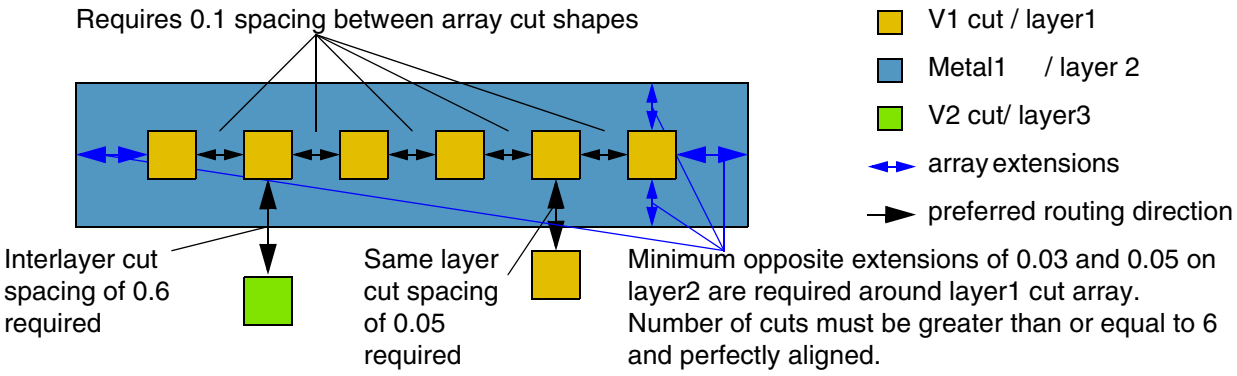
`arraySpace`	Specifies in user units the spacing between this 1D array and other cut classes, or another 1D array if the parallel run length is greater than 0. Type: `Value`
`interViaSpace`	Specifies the required spacing in user units between this 1D array and any different-net cuts on all cut classes on the other cut layer (layer3) of the constraint's layer array. Type: `Value`
`dualExtension`	Specifies in user units the required opposite extensions of the metal layer (layer2) beyond the cut array. Type: `DualValue`

Examples

In the following example:

- Edge-to-edge spacing between six or more perfectly aligned 0.03 x 0.03 cuts (VA) in a 1D array on the V1 cut layer in the preferred routing direction is 0.1.
- Minimum opposite extensions on the `Metal1` layer beyond the array are 0.03 and 0.05.
- Minimum spacing between the cut array and any other cut shape on V1 is 0.5.
- Minimum interlayer spacing between the array and any other V2 cut shape is 0.6.

```
set_constraint_parameter -name cutClass -DualValue {0.03 0.03}
set_constraint_parameter -name numCuts -IntValue 6
set_constraint_parameter -name arraySpace -Value 0.5
set_constraint_parameter -name interViaSpace -Value 0.6
set_constraint_parameter -name dualExtension -DualValue {0.03 0.05}
set_layerarray_constraint -constraint minOneDArrayStructure \
 -layer1 V1 -layer2 Metal1 -layer3 V2 -Value 0.1
```



Related Topics

Miscellaneous Constraints

oaGateOrientation

Specifies the orientation required for building gates on a layer as one of the following: `horizontal`, `vertical`, or `any`. Gates can be built using any orientation if `any` is specified or if this constraint is not set. Optional `width` parameter limits application of the constraint to only those gates that have a width equal to or less than the set value.

oaGateOrientation Quick Reference

Constraint Type	Layer
Value Types	`StringAsIntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

`StringAsIntValue` — String value that is stored as an integer value.

Values:

`anyGateOrientation`	0 (any direction)
`horizontalGateOrientation`	1 (horizontal)
`verticalGateOrientation`	2 (vertical)

Optional Parameter

`width` — Specifies in user units the maximum gate width at which this constraint applies. If not specified, the orientation applies to all gates.

Type: `Value`

Examples

```
set_constraint_parameter -name width -Value 1.2
set_layer_constraint -constraint oaGateOrientation -layer poly1 \
 -StringAsIntValue "horizontalGateOrientation"
```

Related Topics

Miscellaneous Constraints

rectangularShapeDirection

Specifies that shapes on a given layer must be rectangular and in a specified direction. Square shapes would meet both vertical and horizontal directions. You can optionally specify that the direction of the shape is dependent on the shape's width and that the constraint does not apply to metal shapes of a specified size that contain vias on a given layer and of a given size.

This constraint is used with `allowedWidthRange` which specifies a set of discrete widths for the layer. There should also be a corresponding `allowedLengthRange` to specify a minimum length greater than or equal to the minimum width to avoid ambiguity when determining the direction.

rectangularShapeDirection Quick Reference

Constraint Type	Layer
Value Types	`IntValue`, `Int1DTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Types

`IntValue`	Specifies the direction for shapes on the layer. Valid Values: 0 any 1 horizontal 2 vertical
`Int1DTblValue`	Specifies the direction for shapes on the layer, indexed by the width of the shape in user units.

Optional Parameters

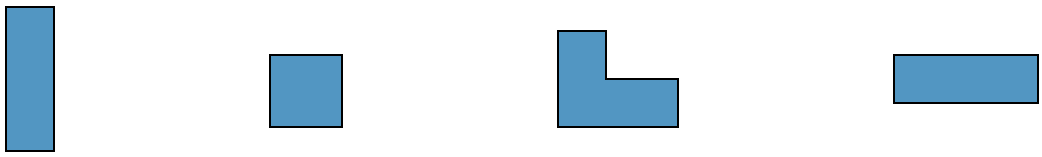
`extendBy`	Specifies that all end-of-line edges between two convex corners must be extended by this distance in user units before checking. This parameter must be specified together with `widthRangeArray`. Type: `Value`
`widthRangeArray`	Specifies that the `extendBy` parameter applies only to edges with lengths (in user units) that lie within in this range. This parameter must be specified together with `extendBy`. Type: `RangeArrayValue`
`exceptExactSize`	Together with `exceptViaSize` and `exceptViaLayer`, specifies that the constraint does not apply to metal shapes that satisfy the following conditions: ■ The size of the metal shape is equal to `exceptExactSize` ■ The metal shape has on it a via with size equal to `exceptViaSize` on layer `exceptViaLayer` ■ The metal shape has one of its edges (in the same direction as the constraint value) aligned to one of the edges of the other metal shapes After removing the metal shapes that satisfy these conditions, the remaining shape must be a either a single rectangle or multiple completely aligned rectangles. Type: `DualValue`
`exceptViaSize`	Specifies the xSpan and ySpan of via that causes a metal shape to be excluded from the constraint. See `exceptExactSize` for more information. Type: `DualValue`
`exceptViaLayer`	Specifies the via layer that causes a metal shape to be excluded from the constraint. See `exceptExactSize` for more information. Type: `LayerValue`

Examples

Example 1: rectangularShapeDirection using IntValue

Specifies that all shapes on layer `Metal1` must be rectangular and vertical.

```
set_layer_constraint -layer Metal1 \
 -constraint rectangularShapeDirection -IntValue 2
```



a) PASS

b) PASS. Square shape is acceptable for horizontal and vertical directions.

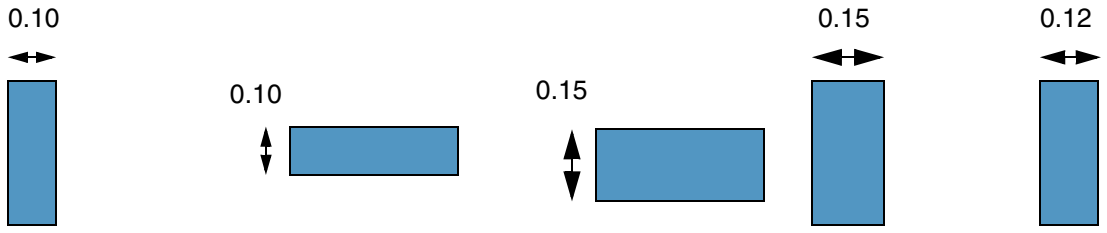
c) FAIL. Shape is not a rectangle.

d) FAIL. Shape is not vertical.

Example 2: rectangularShapeDirection using Int1DTblValue

Specifies that all shapes on layer `Metal1` must be rectangular and shapes that are 0.10 user units wide must be vertical, shapes that are 0.15 user units may be horizontal or vertical.

```
set_layer_constraint -layer Metal1 \
 -constraint rectangularShapeDirection -row_name width \
 -Int1DTblValue {0.10 2 0.15 0} -row_interpolation snap_down \
 -row_extrapolation {snap_up snap_down}
```



a) PASS. Wire is 0.10 wide and vertical.

b) FAIL. Wire is 0.10 wide but is not vertical.

c) PASS. Wire is 0.15 wide and can be in either direction.

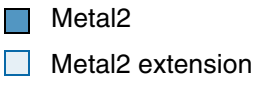
d) PASS. Constraint does not apply for wire width = 0.12.

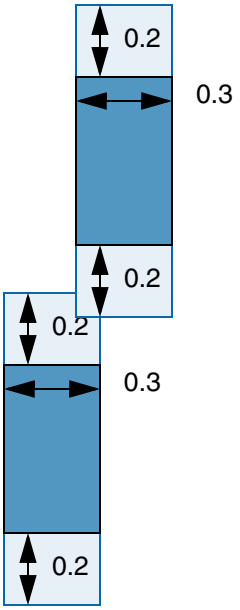
Example 3: rectangularShapeDirection with widthRangeArray and extendBy

Specifies that all end-of-line edges that are less than or equal to 0.3 user units or equal to 0.6 or 0.8 user units in length are to be extended by 0.2 user units. All resulting shapes must be rectangular.

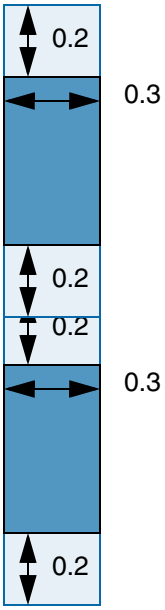
```
set_constraint_parameter -name widthRangeArray \
 -RangeArrayValue {"<=0.3" "0.6" "0.8"}
set_constraint_parameter -name extendBy -Value 0.2
set_layer_constraint -constraint rectangularShapeDirection \
 -layer Metal2 -IntValue 0
```

```
rectangularShapeDirection 0 (any)
  widthRangeArray         {"<=0.3" "0.6" "0.8"}
  extendBy                0.2
```





a) FAIL. Both `Metal2` shapes have end-of-line edges (0.3) in one of the width ranges (<= 0.3), so each edge is extended by 0.2. The resultant shape after extension and merging is not rectangular.



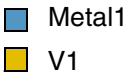
b) PASS. Both `Metal2` shapes have end-of-line edges (0.3) in one of the width ranges (<= 0.3), so each edge is extended by 0.2. The resultant shape after extension and merging is still rectangular.

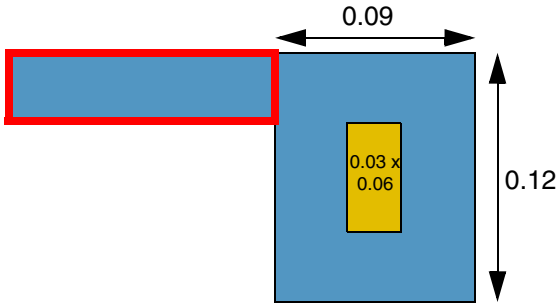
Example 4: rectangularShapeDirection with exceptViaLayer, exceptViaSize, and exceptExactSize

Specifies that any metal shapes on layer `Metal1` must be horizontal rectangles, apart from metal shapes of size 0.09 by 0.12 that contain a via of size 0.03 by 0.06 on layer `V1`.

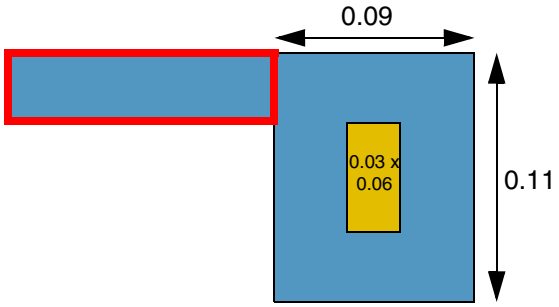
```
set_constraint_parameter -name exceptViaLayer -LayerValue "V1"
set_constraint_parameter -name exceptViaSize -DualValue {0.03 0.06}
set_constraint_parameter -name exceptExactSize -DualValue {0.09 0.12}
set_layer_constraint -layer Metal1 -constraint rectangularShapeDirection \
 -IntValue 1
```

```
rectangularShapeDirection 1
(horizontal)
  exceptViaLayer          V1
  exceptViaSize           {0.03 0.06}
  exceptExactSize         {0.09 0.12}
```

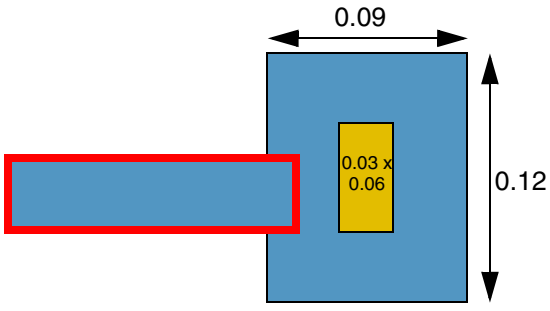




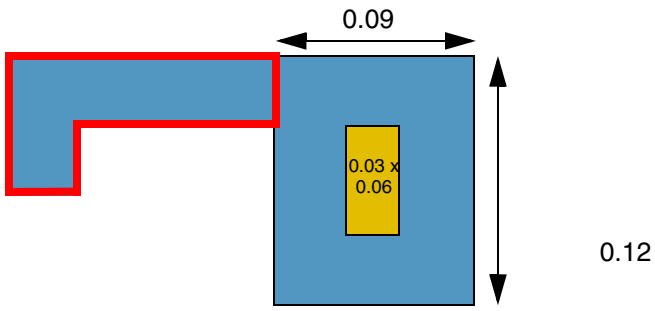
a) PASS. The area (in the red border) is a horizontal rectangle. The constraint applies only to this area and rest of the area is disregarded because it has a via of size 0.03 by 0.06 on layer `V1` and the size of the shape containing the via is 0.09 by 0.12.



b) FAIL. The constraint applies but is not met because the size of metal shape on which the via sits is not 0.09 by 0.12.



c) FAIL. The constraint applies to all metal shapes but is not met because the area (in the red border) does not have any edges that are aligned with the rest of the



d) FAIL. The metal over via is waived due to the exception but the area (in the red border) is not a rectangle.

Related Topics

Miscellaneous Constraints

trimShape

Determines how shapes are formed on a trim metal layer at the line-ends of wires.

Shapes are always formed at the line-ends of wires.

trimShape Quick Reference

Constraint Type	Layer
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

Value Type

IntValue	Specifies the extension model. Valid Values: 0 Shapes extend to the centerline of adjacent tracks (adjacentTrack). 1 Shapes extend by half of the required spacing of the wires (midTrack). 2 Shapes extend by the specified value (edgeExtension) on either side from the center (extension).

Required Parameter

exactWidth	Specifies that the width of the shape formed on the trim metal layer must be equal to this value. Type: Value

Optional Parameters

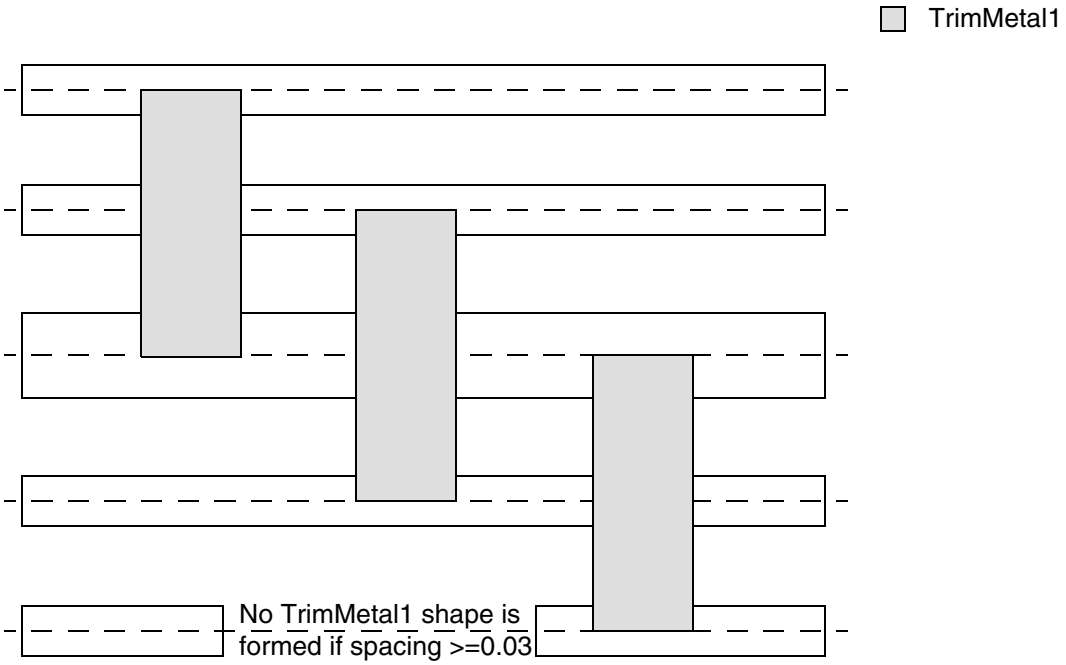
`maxLength`	Specifies that the length of the shape formed on the trim metal layer must be less than or equal to this value. Type: `Value`
`exceptSpacing`	Specifies that no shape is formed on the trim metal layer if the spacing between the line-ends of two wires is greater than or equal to this value. Type: `Value`
`exceptWidth`	Specifies that no shape is formed on the trim metal layer if the width of the line-end of a wire is greater than this value. Type: `Value`
`edgeExtension`	Specifies that the trim metal shape must extend by this value on either side from the center of the metal that it trims. This parameter should be used only when the constraint value is `2` (extension). Type: `Value`
`metalEdge`	Specifies that the trim metal shape must extend from the edges of the metal that it trims. This parameter should be used only when `edgeExtension` is specified. Type: `BoolValue`

Examples

Example 1: trimShape with adjacentTrack

Specifies that the width of each `TrimMetal1` shape must be equal to `0.02` and the `TrimMetal1` shapes must extend to the centerline of adjacent tracks (adjacentTrack). No `TrimMetal1` shape is formed if the spacing between line-ends of two wires is greater than or equal to `0.03`.

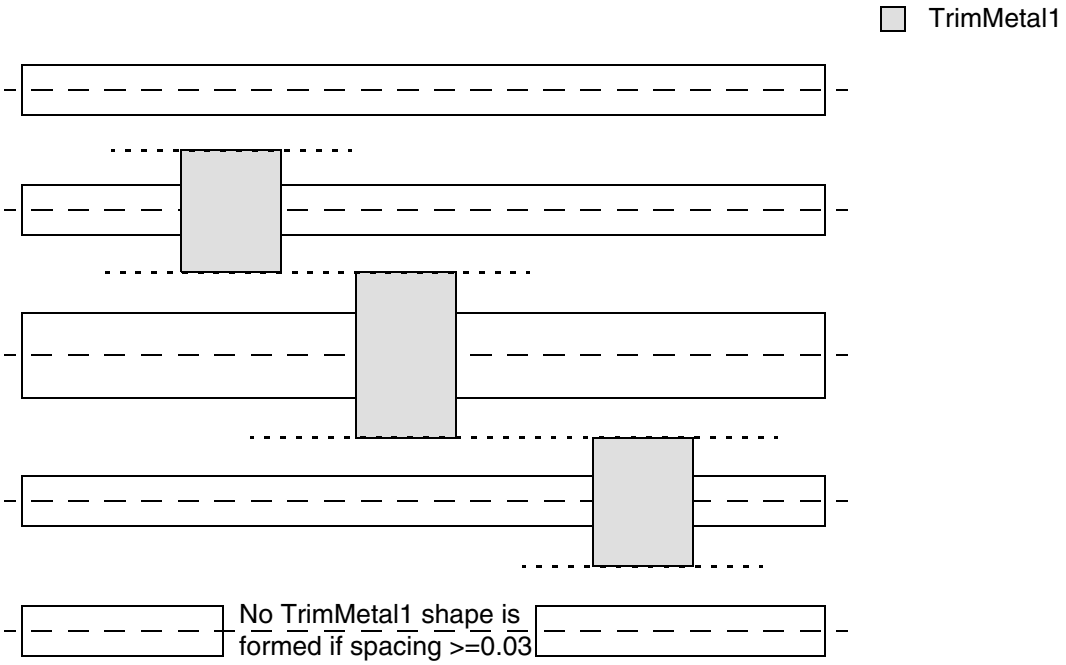
```
set_constraint_parameter -name exactWidth -Value 0.02
set_constraint_parameter -name exactSpacing -Value 0.03
set_layer_constraint -constraint trimShape -layer TrimMetal1 -IntValue 0
```



Example 2: trimShape with midTrack

Specifies that the width of the `TrimMetal1` shapes must be equal to `0.02` and the `TrimMetal1` shapes must extend by half of the required spacing of the wires (midTrack). No `TrimMetal1` shape is formed if the spacing between line-ends of two wires is greater than or equal to `0.03`.

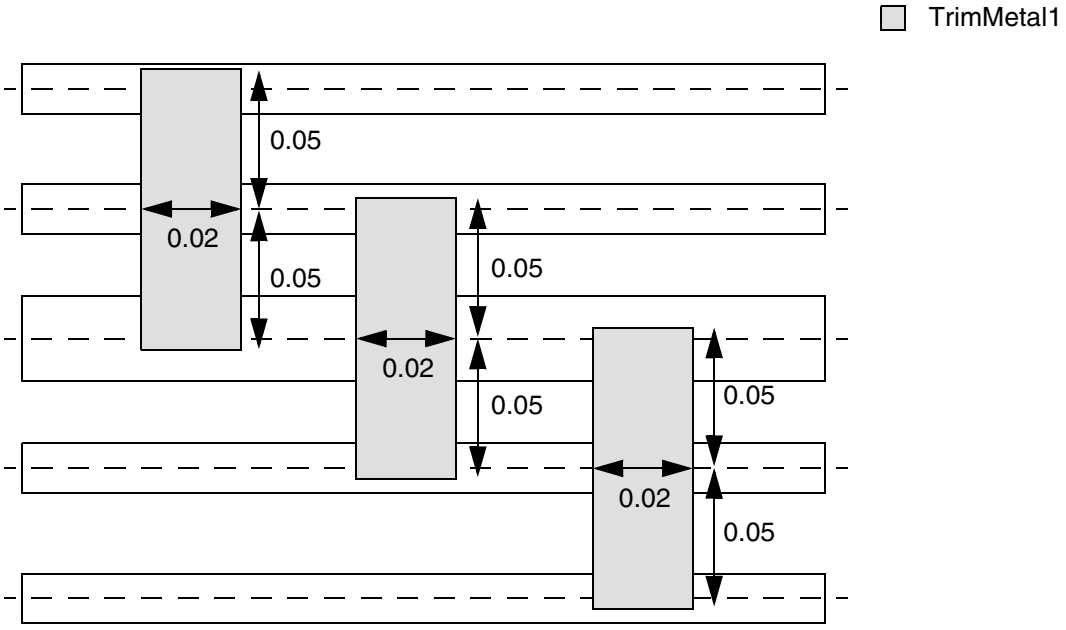
```
set_constraint_parameter -name exactWidth -Value 0.02
set_constraint_parameter -name exactSpacing -Value 0.03
set_layer_constraint -constraint trimShape -layer TrimMetal1 -IntValue 1
```



Example 3: trimShape with edgeExtension and extension

Specifies that the width of each `TrimMetal1` shape must be equal to `0.02` and it must extend by `0.05` on either side from the center of the metal that it trims.

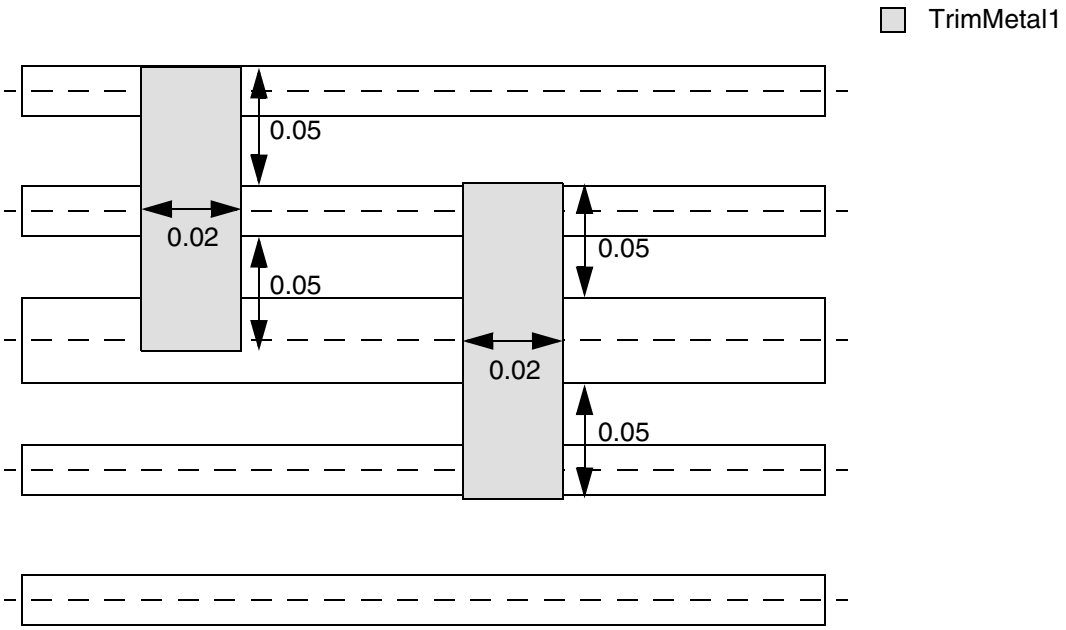
```
set_constraint_parameter -name exactWidth -Value 0.02
set_constraint_parameter -name edgeExtension -Value 0.05
set_layer_constraint -constraint trimShape -layer TrimMetal1 -IntValue 2
```



Example 4: trimShape with edgeExtension, metalEdge, and extension

Specifies that the width of each `TrimMetal1` shape must be equal to `0.02` and it must extend by `0.05` on either side from the metal edges that it trims.

```
set_constraint_parameter -name exactWidth -Value 0.2
set_constraint_parameter -name edgeExtension -Value 0.5
set_constraint_parameter -name metalEdge -BoolValue true
set_layer_constraint -constraint trimShape -layer TrimMetal1 -IntValue 2
```



Related Topics

Miscellaneous Constraints

check_space

useExistingShapesAsShield

Controls whether existing shapes can be used as shields and attract the shielded net to existing shields. This constraint is applied to existing shielded nets.

useExisitingShapesAsShield Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Miscellaneous

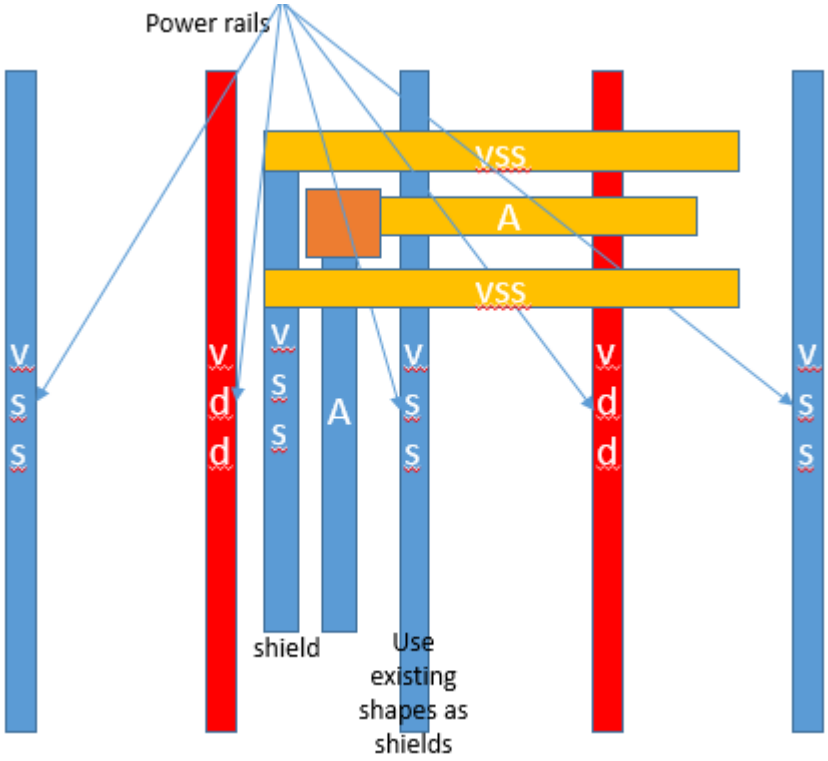
Value Type

BoolValue	When set to `true`, the constraint is enabled.

Examples

Specifies the `useExistingShapesAsShield` constraint to the existing `CG__9` shield constraint group.

```
set_constraint -constraint useExistingShapesAsShield -group CG__9 -BoolValue true
```



Related Topics

Miscellaneous Constraints

11

Mixed-Signal Routing Constraints

This topic lists the mixed-signal routing constraints:

crossTalkNeighborIndex	gapSpace	horizontalSymmetryLine
ignoreShieldingOnLayers	isDriver	lengthPatternAccordion
lengthPatternDangle	lengthPatternEndRun	lengthPatternOff
lengthPatternRWAccordion	lengthPatternTrombone	matchTolerance
maxClusterDistance	maxCouplingDiff	maxCouplingDiffPercent
maxGroundCapDiff	maxGroundCapDiffPercent	maxSingleCoupling
maxSingleCouplingPercent	minCoupling	minCouplingPercent
msConnectSupplyDistance	msMatchPerLayer	msMaxCap
msMaxRes	msMaxWidth	msMinCap
msMinLengthPatternPitch	msMinRes	msSameMask
msShieldStyle	msTolerance	numStrands
routeMaxLength	routeMinLength	shareShields
strandSpacing	strandWidth	tandemLayerAbove
tandemLayerBelow	tandemWidth	verticalSymmetryLine

Related Topics

Manage Constraints

Routing Constraints

Obsolete Mixed-Signal Routing Tcl Constraints

crossTalkNeighborIndex

Associates a cost for routing nets on same-layer parallel wires within a neighborhood which can affect signal integrity by introducing crosstalk. The constraint is supported only as a group-to-group constraint.

crossTalkNeighborIndex Quick Reference

Constraint Type	Simple
Value Types	`IntValue`
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

The `crossTalkNeighborIndex` constraint has an `IntValue` of 0, 1, or 2, where 0 is low cost for good neighbors, and 2 is high cost for groups that should be routed apart.

Parameters

None

Related Topics

Mixed-Signal Routing Constraints

Classifying Neighbor Nets

Avoiding Crosstalk

Half-Shielding Nets

gapSpace

Specifies the spacing required between the signal net and its parallel shields.

Important

This constraint is obsolete. The current method uses `minSpacing` in the `shield` constraint group for a net to specify the spacing between signal nets and their parallel shields.

gapSpace Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`gapSpace` constraints have a `Value` that represents the spacing, in user units, required between signal nets and their shield routes on the layer.

Examples

Sets the spacing on `Metal4` between `net1` and its parallel shields to 0.8.

```
set_layer_constraint -layer Metal4 -constraint gapSpace -Value 0.8 \
  -hardness hard -group single_shield
assign_constraint_group -group single_shield -net net_1
```

Related Topics

Mixed-Signal Routing Constraints

horizontalSymmetryLine

Sets the symmetry line on the Y-axis for a symmetric net pair group.

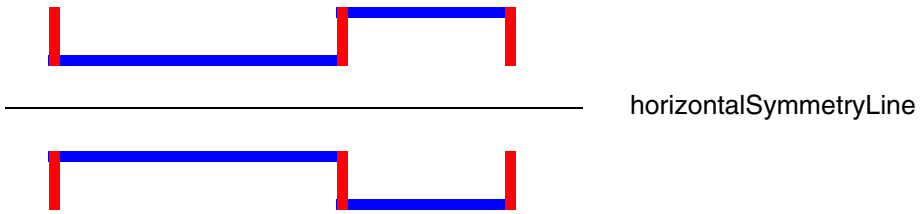
The horizontalSymmetryLine and verticalSymmetryLine constraints are mutually exclusive.

horizontalSymmetryLine Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`horizontalSymmetryLine` constraints have a `Value` that represents the Y-axis coordinate for the symmetry line.



Examples

Sets the symmetry line on the Y-axis, offset 110.5 μm from the origin.

```
set_constraint -constraint horizontalSymmetryLine -Value 110.5
```

Related Topics

Mixed-Signal Routing Constraints

ignoreShieldingOnLayers

Specifies the layers on which shields must not be created.

ignoreShieldingOnLayers Quick Reference

Constraint Type	Simple
Value Types	LayerArrayValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`ignoreShieldingOnLayers` constraints have a `LayerArrayValue` that prevents shields from being routed on the layers in the list.

Examples

This example prevents shielding on layers `Metal2` and `Metal3`.

Format	Example
Tcl	`set_constraint -constraint ignoreShieldingOnLayers \ -LayerArrayValue {Metal2 Metal3}`

Related Topics

Mixed-Signal Routing Constraints

isDriver

Specifies whether an instTerm is a driver. This is useful when making driver-to-sink connections and multiple instTerms are bidirectional.

isDriver Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

The `isDriver` constraints have a `BoolValue` that specifies whether an instTerm is a driver for a net.

Examples

This example assigns the vdd1 term of instance VDDA as a driver.

Format	Example

Tcl	`set instterm [find_inst_term -name vdd1 -instance_name VDDA]`
	`create_constraint_group -name drivers`
	`set_constraint_group -set $instterm -implicit drivers`
	`set_constraint -constraint isDriver -BoolValue true -group drivers`

Related Topics

Mixed-Signal Routing Constraints

lengthPatternAccordion

Determines whether accordion patterns can be used to lengthen wires for matched length routing.

lengthPatternAccordion Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether accordion patterns can be used to lengthen wires. By default, all elongation methods can be used.

Accordion pattern 

Examples

Prevents accordion patterns from being used to lengthen wires.

```
set_constraint -constraint lengthPatternAccordion -BoolValue false
```

Related Topics

Mixed-Signal Routing Constraints

lengthPatternDangle

Specifies a dangle style to use when lengthening wires during performing matched length routing.

lengthPatternDangle Quick Reference

Constraint Type	Simple
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

IntValue

Specifies the dangle style.

By default, dangles are not used for elongation. This constraint must be set to a valid value to enable the use of dangles.

Valid Values:

- 1
  Extends the ends of segments. This can be a preferred method of elongation to avoid wrong-way routing.

Examples

Enables elongation when performing matched length routing by extending ends of segments.

```
set_constraint -constraint lengthPatternDangle -IntValue 1
```

Related Topics

Mixed-Signal Routing Constraints

lengthPatternEndRun

Determines whether end run patterns can be used to lengthen wires for matched length routing.

lengthPatternEndRun Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether end run patterns can be used to lengthen wires. By default, all elongation methods can be used.

End run pattern 

Examples

This example prevents end run patterns from being used to lengthen wires.

```
set_constraint -constraint lengthPatternEndRun -BoolValue false
```

Related Topics

Mixed-Signal Routing Constraints

lengthPatternOff

Determines whether wires can be lengthened on the layer.

lengthPatternOff Quick Reference

Constraint Type	Layer
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether wires can be lengthened on the specific layer.

Examples

This example prevents wires from being lengthened on `Metal2`.

```
set_layer_constraint -constraint lengthPatternOff -layer Metal2 -BoolValue true
```

Related Topics

Mixed-Signal Routing Constraints

lengthPatternRWAccordion

Determines whether wrong-way accordion patterns can be used to lengthen wires for matched length routing.

lengthPatternRWAccordion Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether wrong-way accordion patterns can be used to lengthen wires. By default, all elongation methods can be used.

Wrong way accordion pattern 

Examples

This example permits wrong way accordion patterns to be used to lengthen wires.

```
set_constraint -constraint lengthPatternRWAccordion -BoolValue true
```

Related Topics

Mixed-Signal Routing Constraints

lengthPatternTrombone

Determines whether trombone patterns can be used to lengthen wires for matched length routing.

lengthPatternTrombone Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether trombone patterns can be used to lengthen wires. By default, all elongation methods can be used.

Trombone Pattern

Examples

This example prevents trombone patterns from being used to lengthen wires.

```
set_constraint -constraint lengthPatternTrombone -BoolValue false
```

Related Topics

Mixed-Signal Routing Constraints

matchTolerance

Specifies the absolute tolerance, in user units, when matching net lengths. This constraint only applies to net groups of type `net_match`.

matchTolerance Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`Value`	Specifies the tolerance, in user units, for matching shorter nets in the net group to the length of the longest net. Errors are reported on nets that are shorter than the longest net minus the `matchTolerance` value. If both `matchTolerance` and `msTolerance` are specified, `matchTolerance` is used. However, for interoperability, you should only use msTolerance, which is a percentage value. The default tolerance is two times the pitch or four times the gap space.

Examples

Sets the match tolerance to 0.6.

```
set_constraint -constraint matchTolerance -Value 0.6
```

Related Topics

Mixed-Signal Routing Constraints

matchTolerance

maxClusterDistance

Specifies that pins on multi-pin nets that are less than this distance from each other will be considered as one cluster for strand routing. If not specified, 10 * via-to-via pitch (the maximum value from all valid routing layers) is used. Strand routing can only route two clusters per net. Nets with greater than two clusters will not be routed.

maxClusterDistance Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

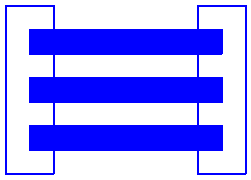
Value Type

`maxClusterDistance` constraints have a `Value` for the maximum distance between pins, in user units, for pins to be clustered together.

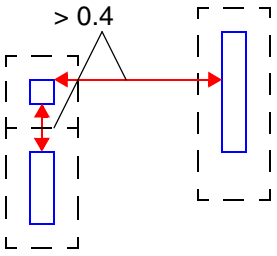
Examples

In this example, pins that are less than or equal to 0.4 μm from another pin on the net will be considered as part of the same cluster for strand routing on the `Metal1` layer.

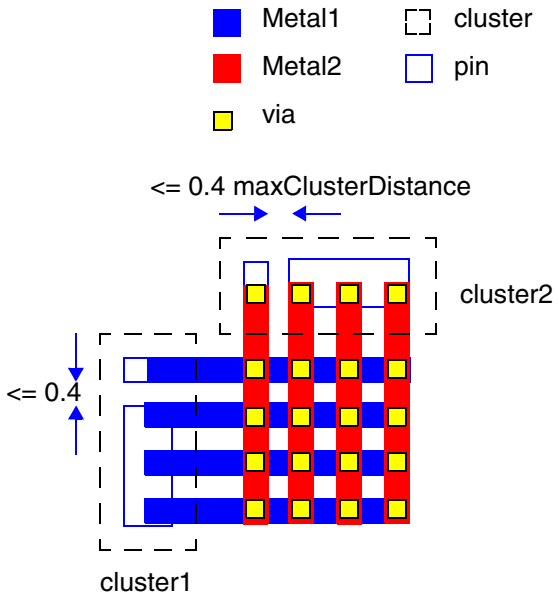
```
set_layer_constraint -constraint maxClusterDistance -layer Metal1 -Value 0.4
```



a) Two-pin net does not require `maxClusterDistance`.



b) Pins are greater than `maxClusterDistance` apart and represent 3 clusters. This net can be routed using the strand router if the `maxClusterDistance` is increased so that the left pins form one cluster.



c) Pins to the left are within `maxClusterDistance` of each other, forming one cluster; pins at the top are within `maxClusterDistance` of each other, forming a second cluster. Strands are routed, connecting the two clusters.

Related Topics

Mixed-Signal Routing Constraints

strand_route

maxCouplingDiff

Specifies the maximum allowed difference in the amount of coupling capacitance from each of the nets within a pair (or group) to all other neighbor nets as an absolute value in Farads.

maxCouplingDiff Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`maxCouplingDiff` constraints have a FltValue that represents the maximum allowed difference, in Farads.

Examples

This example sets the maximum allowed difference in coupling capacitance to 1.0 Farad.

```
set_constraint -constraint maxCouplingDiff -FltValue 1.0
```

Related Topics

Mixed-Signal Routing Constraints

maxCouplingDiffPercent

Specifies the maximum allowed difference in the amount of coupling capacitance from each of the nets within a pair (or group) to all other neighbor nets as a percentage.

maxCouplingDiffPerent Quick Reference

Constraint Type	Simple
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`maxCouplingDiffPercent` constraints have an IntValue that represents the maximum allowed difference as a percentage value.

Examples

This example sets the maximum allowed difference in coupling capacitance, as a percentage, to 20%.

```
set_constraint -constraint maxCouplingCapDiffPercent -IntValue 20
```

Related Topics

Mixed-Signal Routing Constraints

maxGroundCapDiff

Specifies the maximum allowed difference in the amount of ground capacitance between the nets within a pair (or group) as an absolute value in Farads.

maxGroundCapDiff Quick Reference

Constraint Type	Simple
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`maxGroundCapDiff` constraints have a `FltValue` that represents the maximum allowed difference in the amount of ground capacitance as an absolute value in Farads.

Examples

This example sets the maximum allowed difference in ground capacitance to 1.0 Farad.

```
set_constraint -constraint maxGroundCapDiff -FltValue 1.0
```

Related Topics

Mixed-Signal Routing Constraints

maxGroundCapDiffPercent

Specifies the maximum allowed difference in the amount of ground capacitance between the nets within a pair (or group) as a percentage value.

maxGroundCapDiffPercent Quick Reference

Constraint Type	Simple
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`maxGroundCapDiffPercent` constraint `IntValue` that represents the maximum allowed difference in the amount of ground capacitance between the nets as a percentage value.

Examples

This example sets the maximum allowed difference in ground capacitance to 15%.

```
set_constraint -constraint maxGroundCapDiffPercent -IntValue 15
```

Related Topics

Mixed-Signal Routing Constraints

maxSingleCoupling

Specifies the maximum allowed amount of coupling capacitance from all neighbor nets that only couple to one of the nets in a pair as an absolute value in Farads.

maxSingleCoupling Quick Reference

Constraint Type	Simple
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`maxSingleCoupling` constraints have a `FltValue` that specifies the maximum coupling capacitance as an absolute value in Farads.

Examples

This example sets the maximum amount of coupling from all neighbor nets to a single net to 1.0 Farad.

```
set_constraint -constraint maxSingleCoupling -FltValue 1.0
```

Related Topics

Mixed-Signal Routing Constraints

maxSingleCouplingPercent

Specifies the maximum allowed amount of coupling capacitance from all neighbor nets that only couple to one of the nets in a pair as a percentage value.

maxSingleCouplingPercent Quick Reference

Constraint Type	Simple
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`maxSingleCouplingPercent` constraints have an `IntValue` that specifies the maximum allowed amount of coupling capacitance from all neighbor nets that only couple to one of the nets in a pair as a percentage value.

Examples

This example sets the maximum amount of coupling from all neighbor nets to a single net to 15%.

```
set_constraint -constraint maxSingleCouplingPercent -IntValue 15
```

Related Topics

Mixed-Signal Routing Constraints

minCoupling

Sets the minimum coupling capacitance (in Farads) required between the nets.

minCoupling Quick Reference

Constraint Type	Simple
Value Types	`FltValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`minCoupling` constraints have a `FltValue` representing the minimum required coupling capacitance in Farads.

Examples

This example sets the minimum amount of coupling between nets to 0.5 Farad.

```
set_constraint -constraint minCoupling -FltValue 0.5
```

Related Topics

Mixed-Signal Routing Constraints

minCouplingPercent

Sets the minimum coupling capacitance as a percentage required between the nets of a net pair or group.

minCouplingPercent Quick Reference

Constraint Type	Simple
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`minCouplingPercent` constraints have an `IntValue` representing the minimum required coupling capacitance as a percentage.

Examples

This example sets the minimum amount of coupling between nets to 10%.

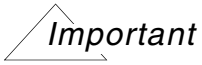
```
set_constraint -constraint minCouplingPercent -IntValue 10
```

Related Topics

Mixed-Signal Routing Constraints

msConnectSupplyDistance

Specifies the maximum distance between ties that must be inserted to tie the new shield wires to their respective shield nets. If this constraint is not specified, only minimal connections are inserted to tie shield wires to their respective nets.



For interoperability with Cadence Innovus™ flows, use this constraint to specify the shield tie distance.

msConnectSupplyDistance Quick Reference

Constraint Type	Simple
Value Types	Value
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

Value	Specifies the maximum distance in user units between shield ties.

Examples

This example sets the shield tie frequency to 2.0 user units.

```
set_constraint -constraint msConnectSupplyDistance -Value 2.0
```

Related Topics

Mixed-Signal Routing Constraints

msMatchPerLayer

Specifies whether length match checks are performed by layer or over the entire length of the net.

msMatchPerLayer Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether length matching is done by layer (`true`), or by total length (`false`, the default), regardless of layer.

Examples

This example specifies that length matching should attempt to match lengths on each layer.

```
set_constraint -constraint msMatchPerLayer -BoolValue true
```

Related Topics

Mixed-Signal Routing Constraints

msMaxCap

Specifies the maximum capacitance for a net.

msMaxCap Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

FltValue	Specifies the maximum capacitance in Farads.

Examples

This example sets the maximum capacitance for nets assigned to constraint group `GB`.

```
set_constraint -group GB -constraint msMaxCap -FltValue 2.0
```

Related Topics

Mixed-Signal Routing Constraints

msMaxRes

Specifies the maximum resistance for a net.

msMaxRes Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

FltValue	Specifies the maximum resistance in ohms.

Examples

This example sets the maximum resistance to 13.0 ohms for nets assigned to constraint group GA.

```
set_constraint -group GA -constraint msMaxRes -FltValue 13.0
```

Related Topics

Mixed-Signal Routing Constraints

msMaxWidth

Specifies the maximum width for a net.

msMaxWidth Quick Reference

Constraint Type	Simple
Value Types	Value
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

Value	Specifies the maximum width in user units.

Examples

This example sets the maximum width to 0.5 user units for nets assigned to constraint group GA.

```
set_constraint -group GA -constraint msMaxWidth -Value 0.5
```

Related Topics

Mixed-Signal Routing Constraints

msMinCap

Specifies the minimum capacitance for a net.

msMinCap Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

FltValue	Specifies the minimum capacitance in Farads.

Examples

This example sets the maximum capacitance to 1.0 Farad for nets assigned to constraint group `GB`.

```
set_constraint -group GB -constraint msMinCap -FltValue 1.0
```

Related Topics

Mixed-Signal Routing Constraints

msMinLengthPatternPitch

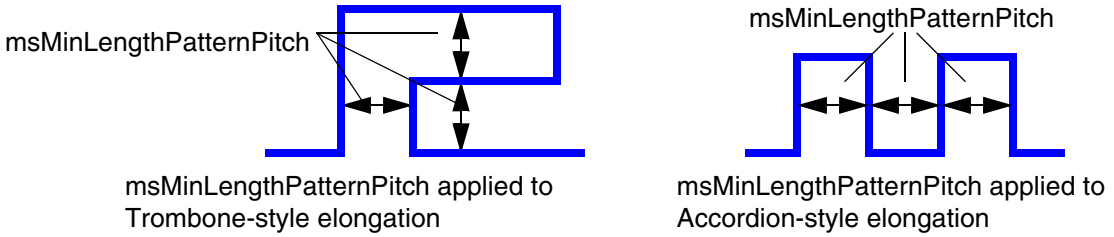
Specifies the minimum same net spacing for elongation patterns. If this constraint is not specified, `minSameNetSpacing` is used for elongation patterns.

msMinLengthPatternPitch Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`Value`	Specifies the minimum spacing in user units.



Examples

This example sets the minimum same net spacing for elongation patterns on `Metal1` to 0.4.

```
set_layer_constraint -constraint msMinLengthPatternPitch -layer Metal1 -Value 0.4
```

Related Topics

Mixed-Signal Routing Constraints

msMinRes

Specifies the minimum resistance for a net.

msMinRes Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

FltValue	Specifies the minimum resistance in ohms.

Examples

This example sets the minimum resistance to 7.0 ohms for nets assigned to constraint group GA.

```
set_constraint -group GA -constraint msMaxRes -FltValue 7.0
```

Related Topics

Mixed-Signal Routing Constraints

msSameMask

Ensures that all the shapes on a given layer are on the same mask. This is a design-level constraint and applies to a net or a net class. This constraint can be assigned as either a soft or a hard constraint.

msSameMask Quick Reference

Constraint Type	Simple
Value Types	BoolValue
Database Types	Design
Scope	design
Category	Mixed Signal Routing

Value Type

BoolValue	Specifies whether shapes on layers must be on the same mask.

Optional Parameter

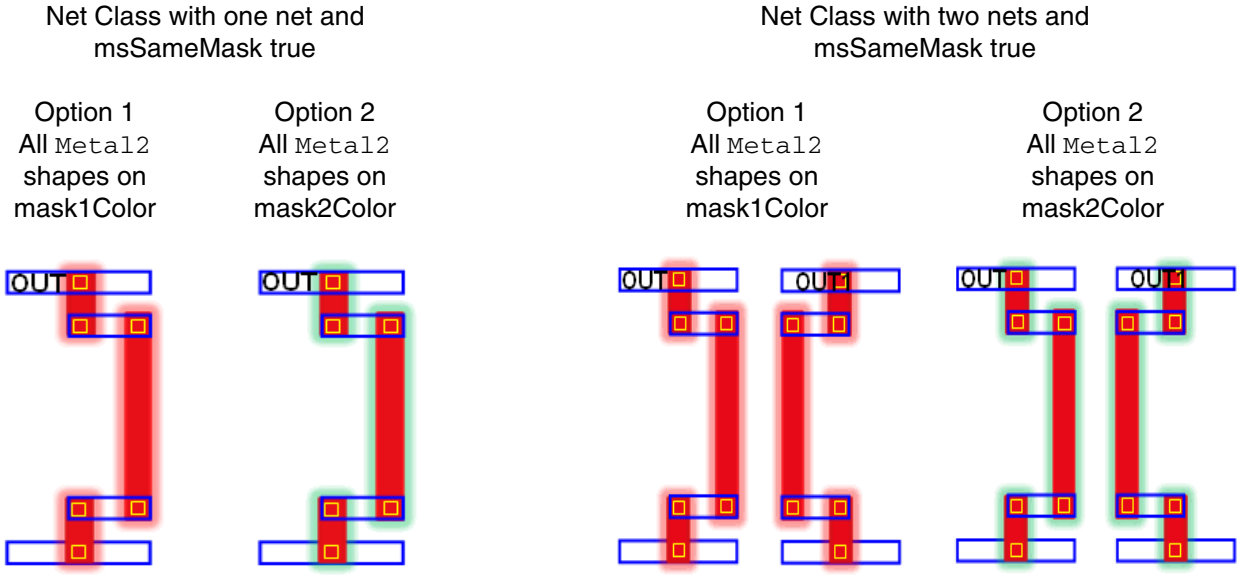
BoolValue	Specifies the name of the mask to be used for the selected shapes (0: any, 1: mask1, 2: mask2, 3: mask3). Type: IntValue 0 any 1 mask1 2 mask2 3 mask3 If there is a shape with this constraint, and the shape's color does not match the color specified by constraint, the checker flags an error.

colorMask (optional)

Examples

The following figure illustrates how shapes on a layer are assigned to the same mask (same color group) when `msSameMask` is `true` for a net or net class.

Illustration of msSameMask



Related Topics

Mixed-Signal Routing Constraints

msShieldStyle

Determines the type of shielding to route around a net.

For interoperability with Cadence Innovus™ flows, you must set this constraint to specify shielding.

msShieldStyle Quick Reference

Constraint Type	Simple
Value Types	IntValue
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

IntValue	Represents a binary-encoded integer, as listed in the table below.

Split	Top	Bottom	Sides	msShieldStyle	Description
0	0	0	0	0	No shields
0	0	0	1	1	Only parallel (sides)
0	0	1	0	2	Only tandem below
0	0	1	1	3	Tandem below and parallel
0	1	0	0	4	Only tandem above
0	1	0	1	5	Tandem above and parallel
0	1	1	0	6	Tandem above and below
0	1	1	1	7	Coaxial (Tandem above and below, parallel)
1	1	1	1	15	Tandem split (Split, tandem above and below, parallel)

Examples

The following example specifies parallel shields only.

```
set_constraint -constraint msShieldStyle -IntValue 1
```

Related Topics

Mixed-Signal Routing Constraints

shield_net

msTolerance

Specifies the allowed percentage tolerance to use when matching routing lengths.

msTolerance Quick Reference

Constraint Type	Simple
Value Types	FltValue
Database Types	Design
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

FltValue	Specifies the percentage tolerance.

Examples

This example sets the tolerance to 10%.

```
set_constraint -constraint msTolerance -FltValue 10
```

Related Topics

Mixed-Signal Routing Constraints

numStrands

Specifies the number of strands that will be used to route between pins for nets in a `net_strand` group.

numStrands Quick Reference

Constraint Type	Simple
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`numStrands` constraints have an `IntValue` for the number of strands.

Examples

The following command specifies that three strands will be used to route pins in `net_strand` groups.

```
set_constraint -constraint numStrands -IntValue 3
```

Related Topics

Mixed-Signal Routing Constraints

strand_route

routeMaxLength

Specifies the maximum route length that is used to fix lengths on individual nets.

routeMaxLength Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`routeMaxLength` constraints have a `Value` that represents the maximum route length, in user units.

Examples

This example set the maximum route length to 50.0.

```
set_constraint -constraint routeMaxLength -Value 50.0
```

Related Topics

Mixed-Signal Routing Constraints

routeMinLength

Specifies the minimum route length that is used to fix lengths on individual nets.

routeMinLength Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`routeMinLength` constraints have a `Value` that represents the minimum route length, in user units.

Examples

This example sets the minimum route length to 23.0.

```
set_constraint -constraint routeMinLength -Value 23.0
```

Related Topics

Mixed-Signal Routing Constraints

shareShields

Specifies whether shield routes can be shared by nets. Usually, the router attempts to individually shield each net. If there is a channel between two nets that is not large enough to add individual shields for each net and `shareShields` is set to `true`, one shield wire will be added between the two nets and shared by the nets.

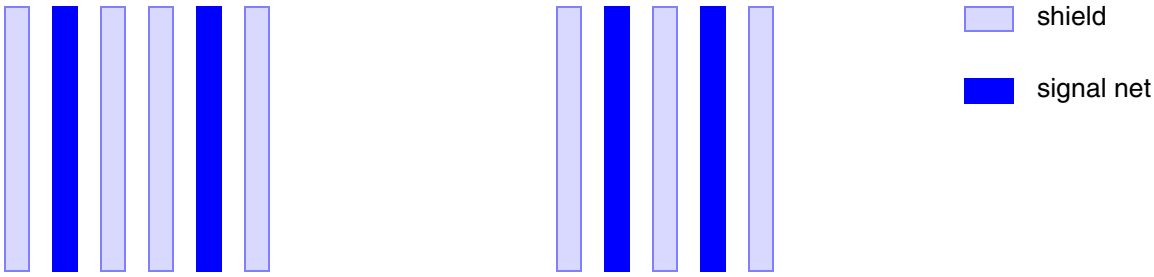
shareShields Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design
Category	Mixed Signal Routing

Value Type

`BoolValue`

Specifies whether shield routes can be shared. If set to `true`, nets can share shield routes. By default and when `false`, each net will be shielded individually. This constraint can be set for a net group or as a global setting on the design route spec (`catenaDesignRule`).



shareShields false

shareShields true

Examples

The following example permits all shielded nets to share shield routes whenever possible.

```
set_constraint -constraint shareShields -BoolValue true
```

The following example permits sharing of shield routes for nets in the `signalNets` set.

```
create_group -set $signalNets -name signal_nets - type group
set_constraint -constraint shareShields -BoolValue true -group signal_nets
```

Related Topics

Mixed-Signal Routing Constraints

strandSpacing

Specifies the required exact spacing between individual wire strands on the given layer.

strandSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`strandSpacing` constraints have a `Value` for the spacing, in user units.

Examples

The following command sets 0.06 μm as the required spacing between individual strands on layer `Metal1` used to connect pins in `net_strand` groups.

```
set_layer_constraint -constraint strandSpacing -layer Metal1 -Value 0.06
```

Related Topics

Mixed-Signal Routing Constraints

strand_route

strandWidth

Specifies the required exact width for individual wire strands on the given layer.

strandWidth Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`strandWidth` constraints have an `Value` for the width, in user units.

Examples

The following command sets 0.03 μm as the required width for individual strands on layer `Metal1` used to connect pins in `net_strand` groups.

```
set_layer_constraint -constraint strandWidth -layer Metal1 -Value 0.03
```

Related Topics

Mixed-Signal Routing Constraints

strand_route

tandemLayerAbove

Specifies the layer above the given signal wire layer that should be used for tandem shields.

tandemLayerAbove Quick Reference

Constraint Type	Layer
Value Types	LayerValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

LayerValue	Specifies the layer above to use for tandem shielding.

Examples

The following example specifies that `Metal6` should be used for shielding `Metal4` signal wires from above.

```
set_layer_constraint -constraint tandemLayerAbove -layer Metal4 -LayerValue Metal6
```

Related Topics

Mixed-Signal Routing Constraints

tandemLayerBelow

Specifies the layer below the given signal wire layer that should be used for tandem shields.

tandemLayerBelow Quick Reference

Constraint Type	Layer
Value Types	LayerValue
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

LayerValue	Specifies the layer below to use for tandem shielding.

Examples

This example specifies that `Metal2` should be used for shielding `Metal4` signal wires from below.

```
set_layer_constraint -constraint tandemLayerBelow -layer Metal4 -LayerValue Metal2
```

Related Topics

Mixed-Signal Routing Constraints

tandemWidth

Specifies the wire width for tandem shield routes on the given layer.

tandemWidth Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`Value`	Specifies the wire width, in user units, for tandem shields on the layer.

Example

This example sets the wire width for tandem shield routes on `Metal5` to 0.6.

```
set_layer_constraint -constraint tandemWidth -layer Metal5 -Value 0.6.
```

Related Topics

Mixed-Signal Routing Constraints

verticalSymmetryLine

Specifies the symmetry line on the x-axis for a symmetric net pair group.

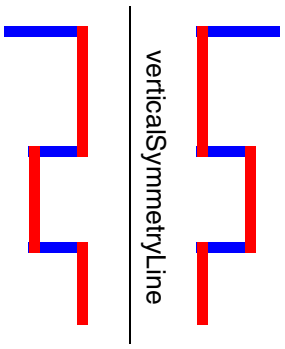
The horizontalSymmetryLine and verticalSymmetryLine constraints are mutually exclusive.

verticalSymmetryLine Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Mixed Signal Routing

Value Type

`verticalSymmetryLine` constraints have a `Value` that specifies the X-axis coordinate for the symmetry line.



Examples

This example sets the symmetry line on the X-axis, offset 55 μm from the origin.

```
set_constraint -constraint verticalSymmetryLine -Value 55
```

Related Topics

Mixed-Signal Routing Constraints

Obsolete Mixed-Signal Routing Tcl Constraints

The following Tcl constraints associated with Virtuoso® Space-based Router are no longer supported in the current release and are ignored if called.

inputTaperRule	Specifies the name of the route spec to use for tapering to input pins. The `inputTaperRule` constraint is obsolete. The current method specifies tapers using constraint groups attached to the objects: ■ `inputtaper` and `outputtaper` constraint group for bit net objects ■ `taper` constraint group for instance terminals, pins, and bit term objects
msMinNumCut	This constraint is obsolete. Use minNumCut instead.
msMinSpacing	This constraint is obsolete. Use minSpacing instead.
msMinWidth	This constraint is obsolete. Use minWidth instead.
msOverhang	This constraint is obsolete. Use tandemWidth instead.
outputTaperRule	Specifies the name of the route spec to use for tapering to output pins. This constraint is obsolete. The current method specifies tapers using constraint groups attached to the objects: ■ `inputtaper` and `outputtaper` constraint group for bit net objects ■ `taper` constraint group for instance terminals, pins, and bit term objects.
shieldRule	Specifies the constraint group from which to get shield parameters (minSpacing and minWidth). This constraint is obsolete. The current method for specifying shields is to create a shield constraint group for the nets that you want to shield. For more information, see Configuring Shield Wires.

shieldWidth	Specifies the width to be applied to parallel shield routes on a layer. This constraint is obsolete. The current method uses minWidth in the shield constraint group for a net to specify the width of its parallel shields. For more information, see Configuring Shield Wires.
taperRule	Specifies the name of the constraint group that contains taper settings. This constraint is obsolete. The current method specifies tapers using constraint groups attached to the objects: ■ `inputtaper` and `outputtaper` constraint group for bit net objects ■ `taper` constraint group for instance terminals, pins, and bit term objects.

12

NumCut Constraints

This topic lists the numCut constraints:

- minNumCut
- minProtrusionNumCut

Related Topics

Tcl Constraint Commands

Manage Constraints

minNumCut

Specifies the minimum number of cuts that a via object or a via instance must contain when connecting two wide wires or a wide wire and a pin. The number of via cuts required depends on the width of the wires.

Note: This constraint can be applied only to layers of material `cut`.

minNumCut Quick Reference

Constraint Type	Layer
Value Types	`Int1DTblValue`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, net group, design, foundry
Category	NumCut
Group Operators	AND, OR

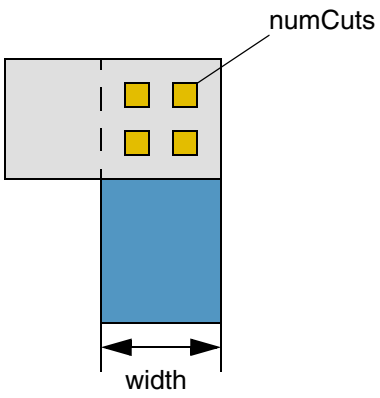
Value Type

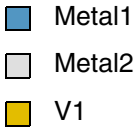
`Int1DTblValue`	The lookup key (`width`) is the width of the connecting metal and the value is the number of cuts required. You must also specify the cut layer name.

For example:

```
set_constraint_parameter -name oaDistanceWithin -Value f_distance
set_constraint_parameter -name metalLayerLocation \
  -StringAsIntValue { metalIsAboveAndBelow | metalIsAbove | metalIsBelow } ]
set_constraint_parameter -name oaArea -Value f_micron2]
```

```
set_layer_constraint -constraint minNumCut -layer cutlayer \
  -Int1DTblValue {{width numCuts}…}
```





The number of cuts must be greater than or equal to the value of this constraint as determined by the width of either of the two layers connected through the cut layer.

Optional Parameters

`cutClass`
Identifies the cut class by width and length. The constraint applies only to cuts class of the given dimensions.

Type: `DualValue`

`oaDistanceWithin`
Specifies the distance at which cuts cannot be counted toward the minimum number of required cuts. Cuts must be closer than the specified distance in order to be counted.

Type: `Value`

`metalLayerLocation`
Identifies whether there must be a metal layers above and/or below the cut layer in order for the constraint to apply.

Type: `StringAsIntValue`

`metalIsAboveAndBelow`	`0` (default)
`metalIsAbove`	`1`
`metalIsBelow`	`2`

`oaArea`
Specifies that the rule applies only when the area of the island is greater than this value, in user units^2, for the given wire widths.

Type: `Value`

`neighborLayerMetal` Specifies the wire that is to be used to determine the required number of cuts.

Type: StringAsIntValue

`upperLowerLayerMetal` `0` (default)
The number of cuts required is determined based on the width of the wider of the two wires.

`upperLayerMetal` `1`
The number of cuts required is determined based on the width of the wire on the layer above.

`lowerLayerMetal` `2`
The number of cuts required is determined based on the width of the wire on the layer below.

`bothUpperLowerLayerMetal` `3`
The number of cuts required is determined based on the width of the narrower of the two wires.

Examples

Establishes a spacing table that sets the minimum number of cuts on `V1` to

- `1` when the width of either of the layers connected to the cut layer is greater than or equal to `0` and less than `0.3`
- `2` when the width of either of the layers connected to the cut layer is greater than or equal to `0.3` and less than `0.6`
- `3` when the width of either of the layers connected to the cut layer is greater than or equal to `0.6` and less than `1.0`
- `5` when the width of either of the layers connected to the cut layer is greater than or equal to `1.0`

For this example, `Metal1` is the layer below `VIA1` and `Metal2` is the layer above `VIA1`.

```
set_constraint_parameter -name oaDistanceWithin -Value 0.500
set_layer_constraint -layer VIA1 -constraint minNumCut \
  -hardness hard -row_name width \
  -Int1DTblValue { 0 1 0.3 2 0.6 3 1.0 5 } \
  -row_interpolation snap_down_inclusive \
  -row_extrapolation {snap_up snap_down}
```

Related Topics

NumCut Constraints

minProtrusionNumCut

Specifies the number of cuts a via must have when the via is placed on a thin wire directly connected to a wide wire or pin. The rule applies only when

- The width of the wide wire is greater than *width*,
- The distance from the wide wire to the first via cut on the protruded piece of metal is less than *distance*, and
- Either the length of the wide wire is greater than *length* or the area of the island (including the wide wire and protrusion) is greater than *area*.

The *width*, *length, distance*, and *area* values are given by the `width`, `length`, `distance`, and `oaArea` parameters.

Note: This constraint can be applied only to layers of material `cut`.

minProtrusionNumCut Quick Reference

Constraint Type	Layer
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	NumCut
Group Operators	AND, OR

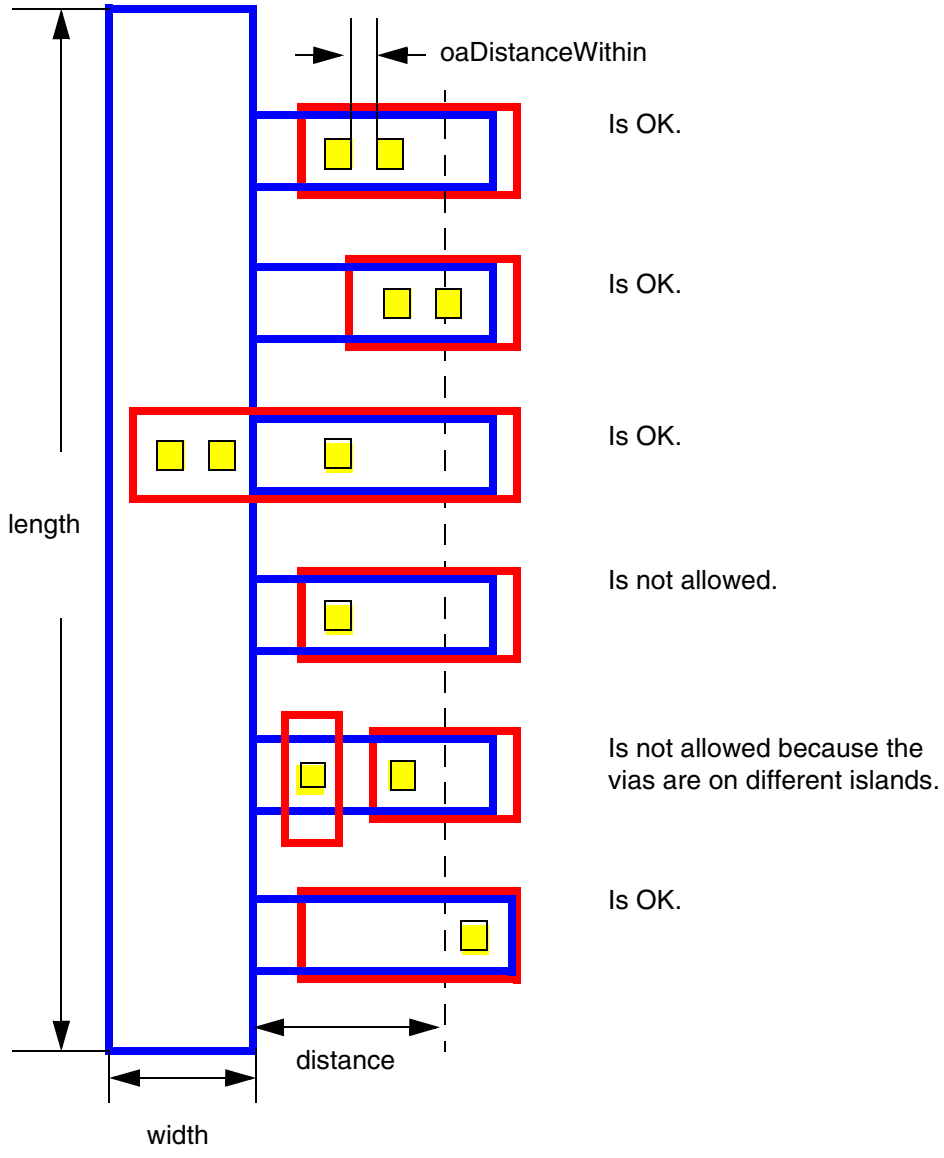
Value Type

`IntValue`	Specifies the number of cuts required. You must also specify the cut layer name.

For example:

```
set_constraint_parameter -name width -Value f_width
set_constraint_parameter -name length -Value f_length
set_constraint_parameter -name distance -Value f_distance
[set_constraint_parameter -name oaDistanceWithin -Value f_within]
set_layer_constraint -constraint minProtrusionNumCut \
 -layer s_layer -IntValue i_numCuts
```

The following figure shows a wide wire with thin wires protruding from it. A minimum of two cuts is required when the *width*, *length*, and *distance* conditions are met. Each protruding wire is evaluated for compliance with the given `minProtrusionNumCut` rule. By default, *distance* is measured from the edge of the wide wire, as shown below.



Required Parameters

`width`	Specifies the width of the wide wire. Type: `Value`
`distance`	Specifies the distance from the wide wire to the first via on the protruded wire Type: `Value`
`length`	Specifies the length of the wide wire. Either `length` or `oaArea` must be given but not both. Type: `Value`
`oaArea`	Specifies the area of the island (including the wide wire and protrusion). Either `length` or `oaArea` must be given but not both. Type: `Value`

Optional Parameters

`oaDistanceWithin`	Specifies the distance at which cuts are not counted. Cuts at a distance equal to or greater than the specified value are not counted. Type: `Value`
`cutClass`	Specifies that the constraint only applies to cut shapes whose width and height are equal to the values in the `DualValue` pair, specified in that order. Cut class applies to 32nm rules. Type: `DualValue`
`lowerLayerMetal`	Specifies whether the constraint applies only to connections from below the cut layer. Type: `BoolValue`
`upperLayerMetal`	Specifies whether the constraint applies only to connections from above the cut layer. Type: `BoolValue`

`centerLineDistance`	If set to `true`, specifies that the distance from the wide wire to the first via cut on the protruded wire is computed from the centerline of the wide wire. The default value is `false`, which means that the distance is measured from the edge of the wide wire. Type: `BoolValue`

Examples

Sets the minimum number of cuts on a protrusion from a wide wire or pin on `V1` to `4` when all of the following are true:

- The distance between the closest cut to the wide wire and the wide wire, edge to edge, is less than 1.8
- The length of the wide wire is greater than 4.5
- The width of the wide wire is greater than 0.99

The cuts must be within 0.5 user units of each other.

```
set_constraint_parameter -name distance -Value 1.800
set_constraint_parameter -name length -Value 4.5
set_constraint_parameter -name oaDistanceWithin -Value 0.500
set_constraint_parameter -name width -Value 0.99
set_layer_constraint -layer V1 -constraint minProtrusionNumCut \
  -hardness hard -IntValue 4
```

Related Topics

NumCut Constraints

13

Overlap Constraints

This topic lists the overlap constraints:

- edgeMustOverlap
- minConcaveCornerOverlap
- minDirectionalOverlap
- minOverlap
- minWireOverlap

Related Topics

Tcl Constraint Commands

Manage Constraints

edgeMustOverlap

Specifies that the edges on layer1 must be fully overlapped by a shape on layer2. Optionally, this may apply only to edges whose lengths are within a certain range and in a certain direction. This constraint requires an overlap, but does not specify the amount of the required overlap. Use the `minDirectionalOverlap` constraint to specify the required overlap.

edgeMustOverlap Quick Reference

Constraint Type	Layer pair
Value Type	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Overlap

Value Type

`BoolValue`	Specifies that edges on layer1 must be fully overlapped by a shape on another layer2.

Optional Parameters

`edgeDirection`	Constraint applies only to layer1 edges in the specified direction. Type: `IntValue` 0 any 1 horizontalEdge 2 verticalEdge
`endOfLineOnly`	Constraint applies only if the layer1 edge is between two convex corners and its length is within the specified `edgeLengthRangeArray`. Type: `BoolValue`
`edgeLengthRangeArray`	

Constraint applies if the layer1 edge length is within this range.

Type: RangeArrayValue, in user units

Examples

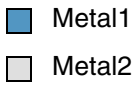
Example 1: edgeMustOverlap with horizontal edgeDirection and edgeLengthRangeArray

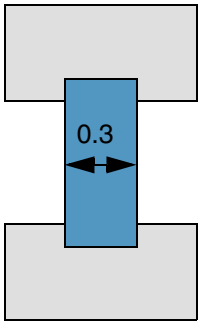
```
set_constraint_parameter -name edgeDirection -IntValue 1
set_constraint_parameter -name edgeLengthRangeArray -RangeArrayValue "<=0.5"
set_layerpair_constraint -layer1 Metal1 -layer2 Metal2 \
  -constraint edgeMustOverlap -BoolValue true
```

Specifies that all horizontal edges on layer `Metal1` that are less than or equal to 0.5 user units in length must be fully overlapped by a shape on layer `Metal2`.

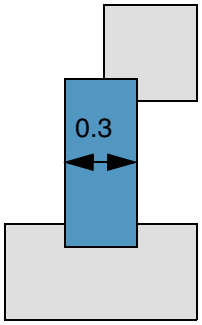
Illustration for edgeMustOverlap with horizontal edgeDirection and edgeLengthRangeArray

```
edgeMustOverlap        true
edgeDirection          1
edgeLengthRangeArray   "<=0.5"
```

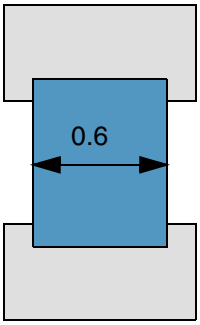




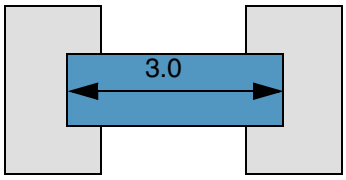
a) PASS. The constraint applies to the top and bottom horizontal `Metal1` edges (<= 0.5) and both are fully overlapped by `Metal2`



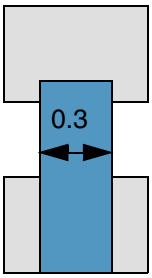
b) FAIL. The constraint applies but the top horizontal `Metal1` edge is not fully overlapped by the `Metal2`



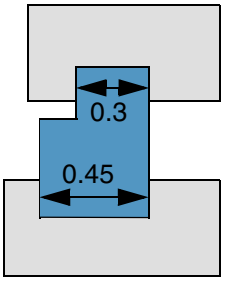
c) Constraint does not apply because the horizontal `Metal1` edge lengths are > 0.5.



d) Constraint does not apply because the horizontal edge lengths are > 0.5.



e) FAIL. The constraint applies and is not met because the bottom `Metal1` edge is not fully overlapped by the `Metal2` shape.



f) FAIL. The constraint applies and is not met because the middle horizontal `Metal1` edge is not fully overlapped by a `Metal2` shape.

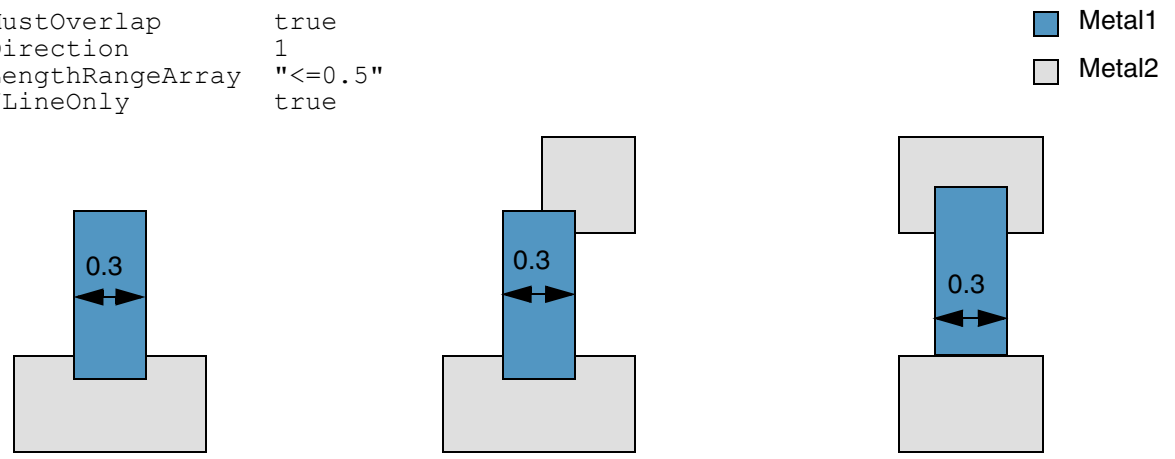
Example 2: edgeMustOverlap with horizontalEdge, endOfLineOnly, and edgeLengthRanges

```
set_constraint_parameter -name edgeDirection -IntValue 1
set_constraint_parameter -name edgeLengthRangeArray -RangeArrayValue "<=0.5"
set_constraint_parameter -name endOfLineOnly -BoolValue true
set_layerpair_constraint -layer1 Metal1 -layer2 Metal2 \
 -constraint edgeMustOverlap -BoolValue true
```

Specifies that all horizontal edges on layer `Metal1` that are between two convex corners and are less than or equal to 0.5 user units in length must be fully overlapped by a shape on layer `Metal2`.

Illustration for edgeMustOverlap with horizontalEdge, endOfLineOnly, and edgeLengthRanges

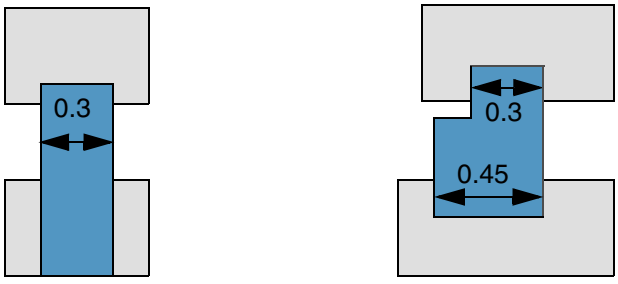
```
edgeMustOverlap        true
edgeDirection          1
edgeLengthRangeArray   "<=0.5"
endOfLineOnly          true
```



a) FAIL. The constraint applies and is not met because the top `Metal1` edge is not overlapped by `Metal2`.

b) FAIL. The constraint applies and is not met because the top `Metal1` edge is not fully overlapped by `Metal2`.

c) FAIL. The constraint applies and is not met because zero overlap is not allowed.



d) FAIL. The constraint applies and is not met because zero overlap is not allowed.

e) PASS. The constraint applies and is met because overlapping only applies to end-of-line edges.

Related Topics

Overlap Constraints

minConcaveCornerOverlap

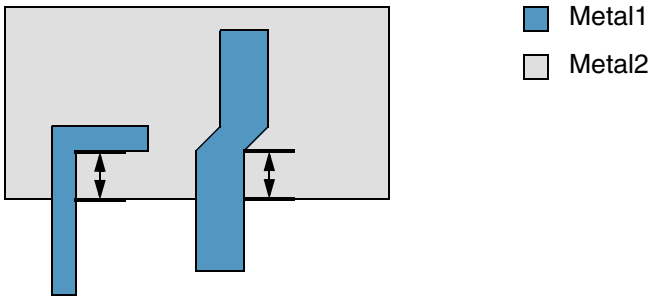
Sets the minimum overlap of a shape on layer2 over an inside corner of a shape on layer1. The minimum inside corner overlap applies to an inside corner created by a geometry at any angle, diagonal, or 90 degrees.

minConcaveCornerOverlap Quick Reference

Constraint Type	Layer pair
Value Type	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Overlap

Value Type

`Value` — Specifies the minimum overlap from the inside corner of a shape on layer1 to the edge of the enclosing shape on layer2.



Examples

The following example sets the minimum inside corner overlap of shapes on layer `Metal2` over shapes with inside corners on layer `Metal1` to `1.0`.

```
set_layerpair_constraint -constraint minConcaveCornerOverlap \
 -layer1 Metal1 -layer2 Metal2 -Value 1.0
```

Related Topics

Overlap Constraints

minDirectionalOverlap

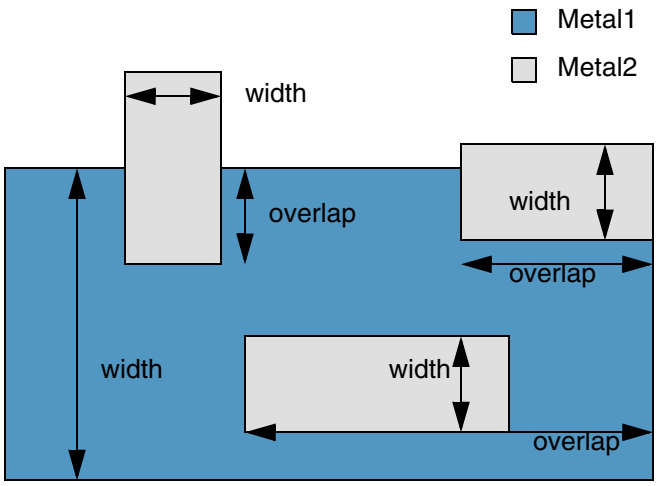
Specifies the required overlap in a given direction between two shapes on different layers. In this case:

- The *width* of a shape is the smaller of its two dimensions.
- The *overlap* between the layer1 and layer2 shapes is measured in the length direction of the layer2 shape, that is, between the outside edge of the layer1 shape and the innermost edge of the layer2 shape.

The constraint can optionally apply only:

- When the layer1 or the layer2 shape is within a specified width range.
- When the layer1 and layer2 shapes are on the same net.
- For overlaps measured in the horizontal or vertical direction. By default, the constraint applies to overlaps in either direction.

Illustration for minDirectionalOverlap



minDirectionalOverlap Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Type	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Overlap

Value Type

`Value`	Specifies the required overlap in user units.

Optional Parameters

`overlapDirection`	Specifies the overlap measurement direction for which the constraint applies. Type: `IntValue` 0 anyOverlap 1 horizontalOverlap 2 verticalOverlap)
`widthRangeArray`	Specifies that the constraint applies only to layer1 shapes with widths in this range. Type: `RangeArrayValue` in user units
`otherWidthRangeArray`	Specifies that the constraint applies only to layer2 shapes with widths in this range. Type: `RangeArrayValue` in user units
`sameNetOnly`	Specifies that the constraint applies only if the layer1 and layer2 shapes are on the same net. The default value of this parameter is `false`. Type: `BoolValue`

Example: minDirectionalOverlap using overlapDirection

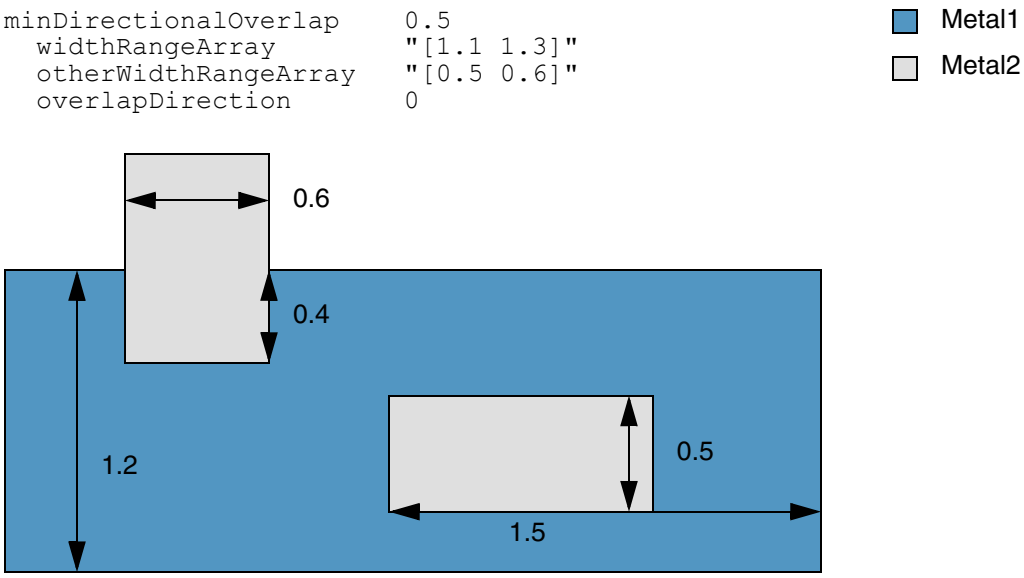
Sets the minimum overlap between the shapes on `Metal1` and `Metal2` to 0.5 user units under the following conditions:

- The width of the `Metal1` shapes is greater than or equal to 1.1 and less than or equal to 1.3.
- The width of the `Metal2` shapes is greater than or equal to 0.5 and less than or equal to 0.6.
- The overlap direction is horizontal and vertical.

```
set_constraint_parameter -name widthRangeArray -RangeArrayValue {"[1.1 1.3]"}
set_constraint_parameter -name otherWidthRangeArray -RangeArrayValue {"[0.5 0.6]"}
set_constraint_parameter -name overlapDirection -IntValue 0
set_layerpair_constraint -layer1 Metal1 -layer2 Metal2 \
  -constraint minDirectionalOverlap -Value 0.5
```

Illustration for minDirectionalOverlap using overlapDirection

```
minDirectionalOverlap    0.5
  widthRangeArray        "[1.1 1.3]"
  otherWidthRangeArray   "[0.5 0.6]"
  overlapDirection       0
```



a) FAIL. The constraint applies because all of the shapes have widths within the values given by widthRangeArray (`Metal1`) and otherWidthRangeArray (L). The constraint value (0.5) is met for the bottom-right `Metal2` shape, measured in the horizontal direction, but fails in the vertical overlap measurement (0.4) for the top-left `Metal2` shape. If the overlapDirection was horizontalOverlap, the constraint would not apply to the top-left `Metal2` shape and the constraint would be satisfied.

Related Topics

Overlap Constraints

minDirectionalOverlap

minOverlap

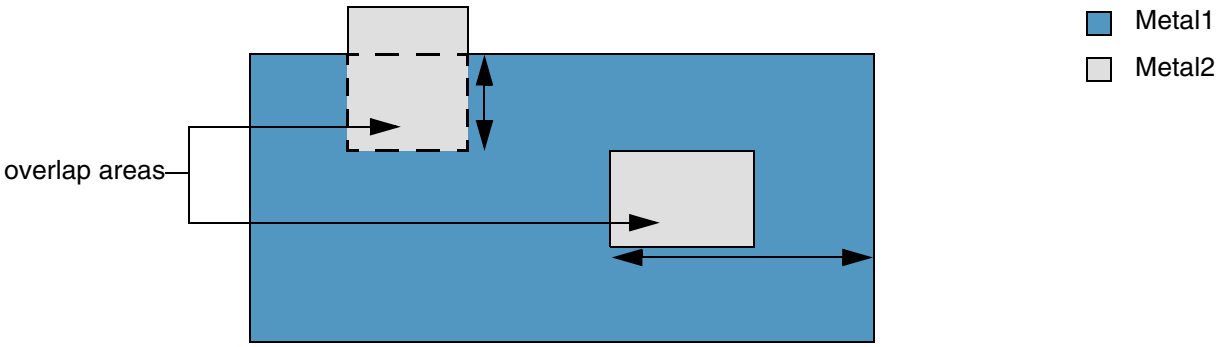
Sets the minimum distance by which a shape on layer1 must overlap a shape on layer2.

minOverlap Quick Reference

Constraint Type	Layer pair
Value Type	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Overlap

Value Type

Value	Represents the minimum overlap in user units.



Any overlap of `Metal1` over `Metal2` must be greater than or equal to the value specified by this constraint in each direction.

Examples

Sets the minimum overlap to the minimum distance between the inside and outside edges of shapes on `Metal1` over shapes on `Metal2` to `0.75`.

```
set_layerpair_constraint -constraint minOverlap \
 -layer1 Metal1 -layer2 Metal2 -Value 0.75
```

Related Topics

Overlap Constraints

minWireOverlap

Applies when two wires on layer1 and layer2 overlap in the same direction. The constraint also specifies the minimum overlap in the length direction along with an optional alignment type. This constraint applies to rectangular shapes only; nonrectangular shapes are ignored.

minWireOverlap Quick Reference

Constraint Type	Layer pair
Value Types	`Value`, `OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Overlap

Value Types

`Value`	Specifies the minimum overlap in user units in the length direction.
`OneDTblValue`	Specifies the minimum overlap in user units in the length direction, indexed by the width of the layer1 wire.

Optional Parameters

`sameDirection`	Specifies that overlapping layer1 and layer2 wires must have the same direction. When `false` (the default), specifies that the constraint does not apply to overlapping perpendicular layer1 and layer2 wires. Type: `BoolValue`
`wireAlignment`	Requires a specific wire alignment when layer1 and layer2 overlap. Type: `IntValue` 0 any (no width or alignment requirements) 1 centerLine (centerlines of both layers must be aligned) 2 sameWidth (wires must have same width and centerlines aligned)

Examples

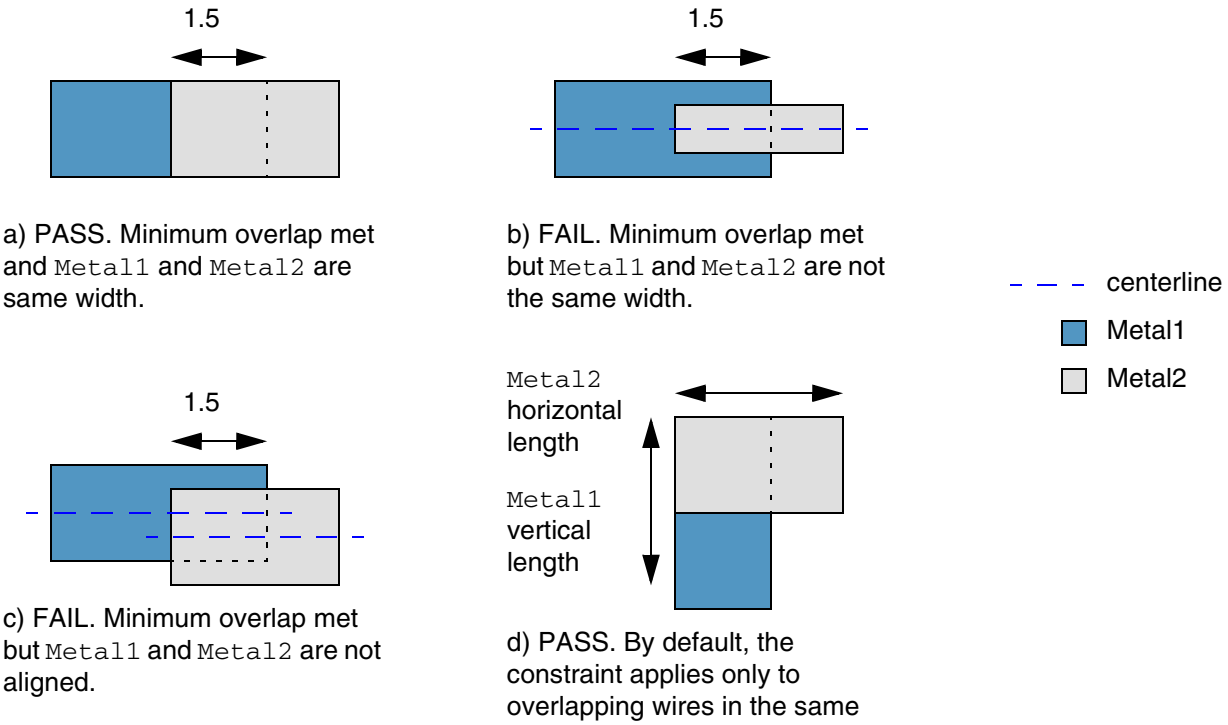
- Example 1: minWireOverlap with wireAlignment=sameWidth
- Example 2: minWireOverlap with wireAlignment=centerLine and sameDirection=true

Example 1: minWireOverlap with wireAlignment=sameWidth

```
set_constraint_parameter -name wireAlignment -IntValue 2
set_layerpair_constraint -constraint minWireOverlap \
 -layer1 Metal1 -layer2 Metal2 -Value 1.5
```

Specifies that overlapping `Metal1` and `Metal2` shapes must have the same width, and their centerlines must be aligned with a minimum overlap of 1.5 user units.

Illustration for minWireOverlap with wireAlignment=sameWidth

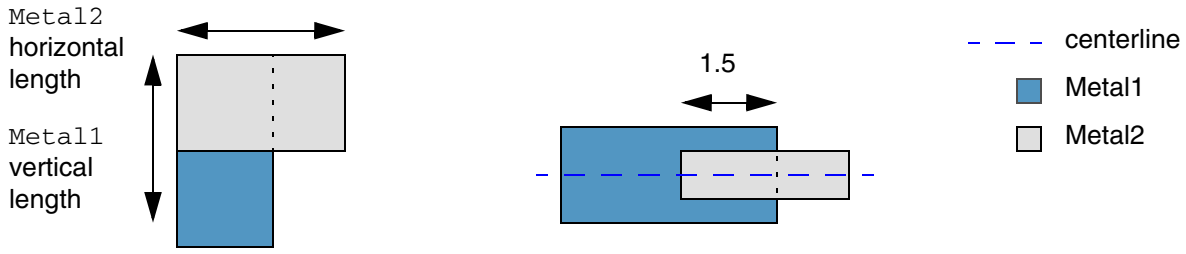


Example 2: minWireOverlap with wireAlignment=centerLine and sameDirection=true

```
set_constraint_parameter -name wireAlignment -IntValue 1
set_constraint_parameter -name sameDirection -BoolValue true
set_layerpair_constraint -constraint minWireOverlap \
 -layer1 Metal1 -layer2 Metal2 -Value 1.5
```

Specifies that overlapping `Metal1` and `Metal2` shapes must have the same direction, and their centerlines must be aligned with a minimum overlap of 1.5 user units.

Illustration for minWireOverlap with wireAlignment=centerLine and sameDirection=true



a) FAIL. Overlapping wires are not in the same direction. If sameDirection is `false` or not specified, there would be no violation because the constraint would not apply.

b) PASS. Minimum overlap met. Wires are in the same direction with centerlines aligned.

Related Topics

Overlap Constraints

minWireOverlap

14

Placement and Alignment Constraints

This topic lists the placement and alignment constraints:

- horizontalPlacementGridOffset
- horizontalPlacementGridPitch
- keepAlignedShapeAndBoundary
- keepAlignedShapes
- verticalPlacementGridOffset
- verticalPlacementGridPitch

Related Topics

Tcl Constraint Commands

Manage Constraints

horizontalPlacementGridOffset

Sets the offset, in user units on the X axis from the origin, for the start of the vertical placement grid.

horizontalPlacementGridOffset Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Placement and Alignment

Value Type

`horizontalPlacementGridOffset` constraints have a `Value` that represents the offset on the X axis from the origin for the vertical placement grid.

Related Topics

Placement and Alignment Constraints

horizontalPlacementGridPitch

Sets the spacing, edge-to-edge and in user units, between vertical placement grids.

horizontalPlacementGridPitch Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Placement and Alignment

Value Type

`horizontalPlacementGridPitch` constraints have a Value that represents the spacing between vertical placement grids.

Related Topics

Placement and Alignment Constraints

keepAlignedShapeAndBoundary

Specifies whether shared edges of shapes and boundaries must remain shared on a layer.

keepAlignedShapeAndBoundary Quick Reference

Constraint Type	Layer
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Placement and Alignment

Value Type

`keepAlignedShapeAndBoundary` constraints have a `BoolValue`.

Related Topics

Placement and Alignment Constraints

keepAlignedShapes

Specifies whether shapes on two layers that have coincident edges must maintain coincident edges.

keepAlignedShapes Quick Reference

Constraint Type	Layer pair
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Placement and Alignment

Value Type

`keepAlignedShapes` constraints have a `BoolValue`. When set to true, coincident edges of shapes on both layers must remain coincident. The length of the shared edge can change as long as the shapes maintain at least one shared point on the shared edge.

Parameter

- `alignmentType` (`IntValue`, optional) specifies the alignment of the shapes as one of the following:

Value	Alignment
0	The coincident edges of the shape on one layer must be **inside** of the shape on the second layer.
1	The coincident edges of the shape on one layer must be **outside** of the shape on the second layer.
2	The coincident edges of the shape on one layer must be either **inside or outside** of the shape on the second layer.

The length of the shared edge can change as long as the shapes maintain at least one shared point on the shared edge.

Related Topics

Placement and Alignment Constraints

verticalPlacementGridOffset

Specifies a vertical placement grid offset for placing instances, relative to the origin of the design.

verticalPlacementGridOffset Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Placement and Alignment

Value Type

`verticalPlacementGridOffset` constraints have a `Value` that represents the vertical offset, in user units, relative to the origin.

Related Topics

Placement and Alignment Constraints

verticalPlacementGridPitch

Specifies an explicit vertical placement grid pitch for placing instances. The grid is specified by the placement grid offset from the design's origin.

verticalPlacementGridPitch Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Placement and Alignment

Value Type

`verticalPlacementGridPitch` constraints have a `Value` that represents the vertical placement grid pitch, in user units.

Related Topics

Placement and Alignment Constraints

15

Routing Constraints

This topic lists the routing constraints.

135RouteGridOffset	135RouteGridPitch	45RouteGridOffset
45RouteGridPitch	clusterDistance	cutClassPreference
defaultHorizontalRouteGridOffset	defaultHorizontalRouteGridPitch	defaultMfgGrid
defaultVerticalRouteGridOffset	defaultVerticalRouteGridPitch	extendedValidRoutingVias
horizontalRouteGridOffset	horizontalRouteGridPitch	inlineViaPreferred
layerHeight	layerThickness	limitRoutingLayers
limitViaThruLayers	maxPickupDistanceAllowed	maxRoutingDistanceAllowed
maxTaperWindow	mfgGrid	minTaperWindow
nearFarPercentage	oaDefault135RouteGridOffset	oaDefault135RouteGridPitch
oaDefault45RouteGridOffset	oaDefault45RouteGridPitch	oaPreferredRoutingDirection
oaTaperHalo	offset	orthogonalSnappingLayer
planarTapAllowed	preferredExtensionDirection	preferredViaOrigin
routeOnGrid	taperToFirstVia	TJunctionAllowed
validRoutingLayers	validRoutingVias	validWireEditorVias
verticalRouteGridOffset	verticalRouteGridPitch	viaTapAllowed
wireExtent	wrongWayOK	

Related Topics

Mixed-Signal Routing Constraints

135RouteGridOffset

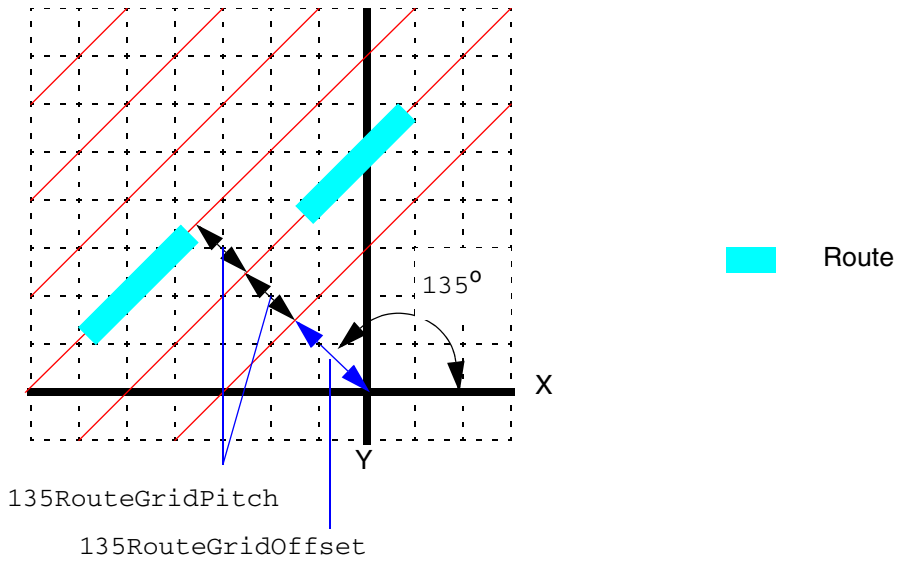
Specifies the offset from the origin for the start of the left diagonal routing grid for the layer. The left diagonal axis is found by rotating the horizontal axis 135 degrees counterclockwise.

135RouteGridOffset Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`135RouteGridOffset` constraints have a `Value` that represents the offset from the origin as a diagonal distance in user units.



Examples

This example set the left diagonal routing grid offset on `Metal3` to 1.2 user units.

Format	Example
Tcl	`set_layer_constraint -constraint 135RouteGridOffset \` `-layer Metal3 -Value 1.2`

Virtuoso	`routingGrids(`
	`(leftDiagOffset "Metal3" 1.2)`
	`)`

Related Topics

Routing Constraints

135RouteGridPitch

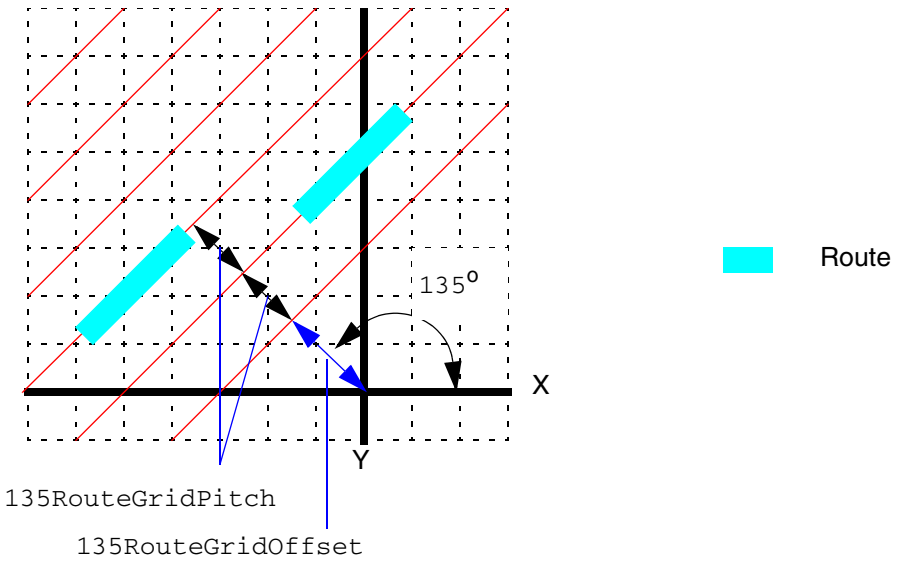
Sets the 135-degree spacing, edge-to-edge and in user units, between diagonal, 45-degree routing grids for the layer.

135RouteGridPitch Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`135RouteGridPitch` constraints have a `Value` that represents the pitch as a diagonal distance in user units.



Examples

This example sets the left diagonal routing grid pitch on `Metal3` to `1.0` user units.

Format	Example
Tcl	`set_layer_constraint -constraint 135RouteGridPitch \` `-layer Metal3 -Value 1.0`

Virtuoso	`routingGrids(`
	`(leftDiagPitch "Metal3" 1.0)`
	`)`

Related Topics

Routing Constraints

45RouteGridOffset

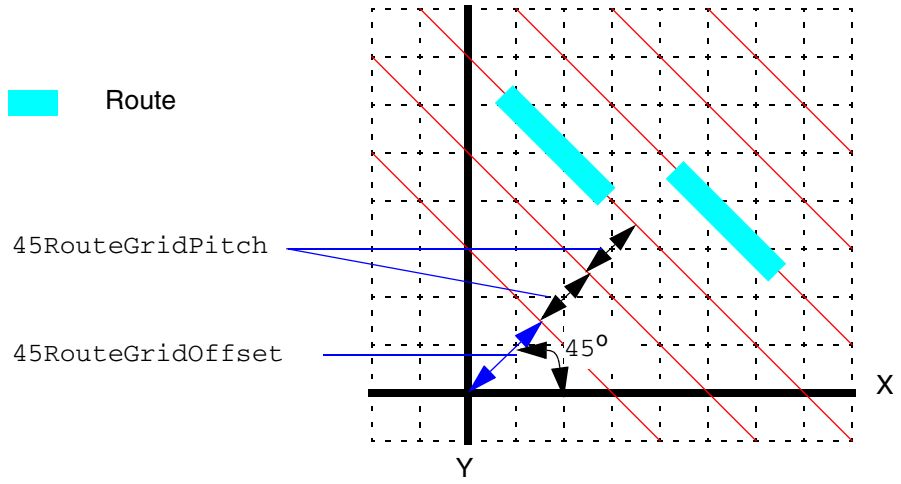
Specifies the offset from the origin for the start of the right diagonal routing grid for the layer. The right diagonal axis is found by rotating the horizontal axis 45 degrees counterclockwise.

45RouteGridOffset Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`45RouteGridOffset` constraints have a `Value` that represents the offset from the origin as a diagonal distance in user units.



Examples

This example set the right diagonal routing grid offset on `Metal3` to 1.2 user units.

Format	Example
Tcl	`set_layer_constraint -constraint 45RouteGridOffset \` `-layer Metal3 -Value 1.2`

Virtuoso	`routingGrids(`
	`(rightDiagOffset "Metal3" 1.2)`
	`)`

Related Topics

Routing Constraints

45RouteGridPitch

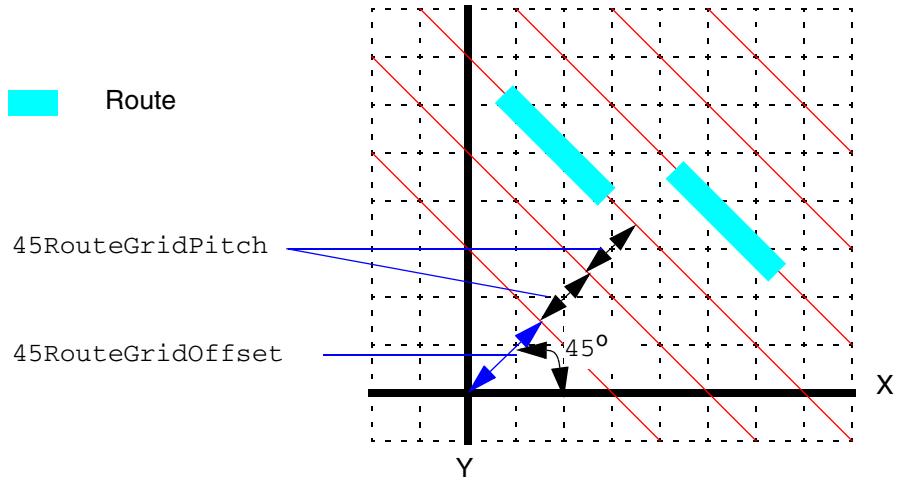
Sets the 45-degree spacing, edge-to-edge and in user units, between diagonal, 135-degree routing grids for the layer.

45RouteGridPitch Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`45RouteGridPitch` constraints have a `Value` that represents the pitch as a diagonal distance in user units.



Examples

This example sets the right diagonal routing grid pitch on `Metal3` to `1.0` user units.

Format	Example
Tcl	`set_layer_constraint -constraint 45RouteGridPitch \` `-layer Metal3 -Value 1.0`

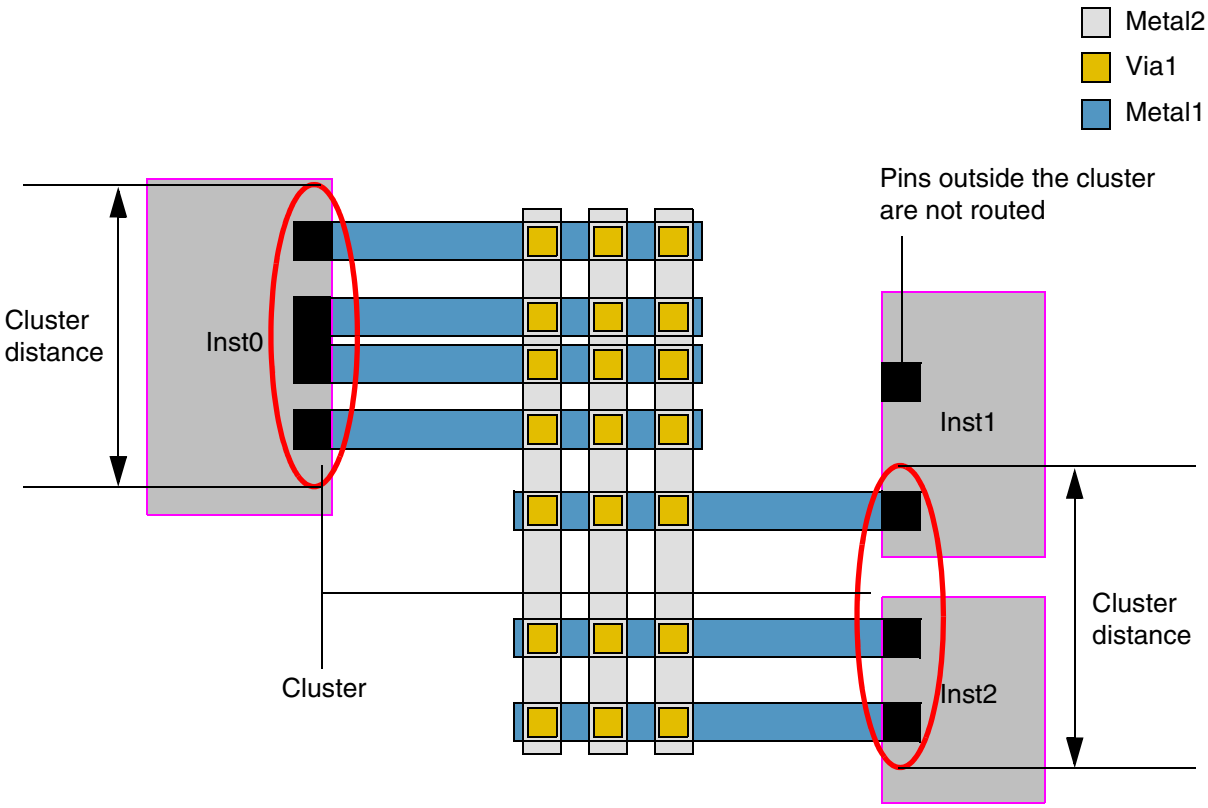
Virtuoso	`routingGrids(`
	`(rightDiagPitch "Metal3" 1.0)`
	`)`

Related Topics

Routing Constraints

clusterDistance

Specifies that pins less than or equal to this distance from each other on a multi-pin net are clustered together for strand routing.



clusterDistance Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`Value`	Specifies the cluster distance, in user units.

Required Parameters

None

Optional Parameters

None

Examples

Specifies that pins spaced less than or equal to 0.06 apart on a multi-pin net are clustered together for strand routing.

```
set_constraint -constraint clusterDistance -Value 0.06
```

Related Topics

Routing Constraints

strand_route

cutClassPreference

Specifies the cut class preferences to be used by the router. If the constraint does not exist, all vias by default are equally preferred. If the constraint exists, but a cut class or a set of cut classes does not exist in the list, those cut classes are less preferred than those in the list.

cutClassPreference Quick Reference

Constraint Type	Layer
Value Types	ValueArrayValue
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction
Group Operators	OR

Value Type

ValueArrayValue	Specifies a list of pairs of values that represent cut class dimensions. The values are specified in order of preference.

Required Parameters

None

Optional Parameters

None

Examples

```
create_constraint_group -name cutClassgp -opType or -db tech
add_constraint_group -subGroupName cutClassgp -groupType foundry

set_constraint_parameter -name className -StringValue Vx
set_constraint_parameter -name numCuts -IntValue 1
set_layer_constraint -constraint cutClass -layer Via1 -create true \
  -group cutClassgp -DualValue {0.5 0.5}

set_constraint_parameter -name className -StringValue VxBar
set_constraint_parameter -name numCuts -IntValue 2
```

```
set_layer_constraint -constraint cutClass -layer Via1 -create true \
  -group cutClassgp -DualValue {0.5 1.0}

set_constraint_parameter -name className -StringValue VxLrg
set_constraint_parameter -name numCuts -IntValue 4
set_layer_constraint -constraint cutClass -layer Via1 -create true \
  -group cutClassgp -DualValue {1.0 1.0}
```

Creates the following three cut classes:

- `Vx 0.5 0.5`
- `VxBar 0.5 1.0`
- `VxLrg 1.0 1.0`

```
set_layer_constraint -constraint cutClassPreference -layer Via1 -create true \
  -group cutClassgp -ValueArrayValue {1.0 1.0}
```

Uses only `VxLrg` for routing.

```
set_layer_constraint -constraint cutClassPreference -layer Via1 -create true \
  -group cutClassgp -ValueArrayValue {1.0 1.0 0.5 1.0}
```

Causes `VxLrg` to be preferred over `VxBar`.

```
set_layer_constraint -constraint cutClassPreference -layer Via1 -create true \
  -group cutClassgp -ValueArrayValue {0.5 1.0 0.5 0.5}
```

Causes `VxBar` to be preferred over `Vx`.

Related Topics

Routing Constraints

defaultHorizontalRouteGridOffset

Specifies the default value for the horizontal route grid offset if a layer-based value is not set.

defaultHorizontalRouteGridOffset Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`defaultHorizontalRouteGridOffset` constraints have a `Value` that represents the default horizontal route grid offset, in user units, from the origin.

Format	Example
Tcl	`set_constraint -constraint defaultHorizontalRouteGridOffset \` `  -Value f_offset`

Examples

The following example sets the default horizontal route grid offset to 0.8.

Format	Example
Tcl	`set_constraint -constraint defaultHorizontalRouteGridoffset \` `  -Value 0.8`

Related Topics

Routing Constraints

defaultHorizontalRouteGridPitch

Specifies the default value for the horizontal route grid pitch if a layer-based value is not set.

defaultHorizontalRouteGridPitch Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`defaultHorizontalRouteGridPitch` constraints have a `Value` that represents the default horizontal route grid pitch in user units from the origin.

Format	Example
Tcl	`set_constraint -constraint defaultHorizontalRouteGridPitch \` `  -Value f_pitch`

Examples

The following example sets the default horizontal route grid pitch to 1.0.

Format	Example
Tcl	`set_constraint -constraint defaultHorizontalRouteGridPitch \` `  -Value 1.0`

Related Topics

Routing Constraints

defaultMfgGrid

Specifies the default manufacturing grid for layers for which it is not explicitly specified. The smallest value for `mfgGrid` is used; if none is found, a value of 1 is used.

defaultMfgGrid Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	derived, internal only
Scope	design
Category	Routing

Value Type

`defaultMfgGrid` constraints have a `Value` that specifies the default manufacturing grid.

Related Topics

Routing Constraints

defaultVerticalRouteGridOffset

Specifies the default value for the vertical route grid offset if a layer-based value is not set.

defaultVerticalRouteGridOffset Quick Reference

Constraint Type	Simple
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`defaultVerticalRouteGridOffset` constraints have a Value that specifies the default vertical route grid offset, in user units from the origin.

Examples

This example set the default vertical route grid offset to 1.0.

Format	Example
Tcl	`set_constraint -constraint defaultVerticalRouteGridOffset \ -Value 1.0`

Related Topics

Routing Constraints

defaultVerticalRouteGridPitch

Specifies the default value, in user units, for the vertical route grid pitch if a layer-based value is not set.

defaultVerticalRouteGridPitch Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`defaultVerticalRouteGridPitch` constraints have a `Value` that represents the default vertical route grid pitch, in user units.

Format	Example
Tcl	`set_constraint -constraint defaultVerticalRouteGridPitch \` `  -Value f_offset`

Examples

This example sets the default vertical route grid pitch to 1.2.

Format	Example
Tcl	`set_constraint -constraint defaultVerticalRouteGridPitch \` `  -Value 1.2`

Related Topics

Routing Constraints

extendedValidRoutingVias

Specifies the list of vias that can be used for routing. These vias are derived from the validRoutingVias constraint with additions from the create_derived_vias command. If the `extendedValidRoutingVias` constraint exists, it is used for routing, instead of using validRoutingVias.

extendedValidRoutingVias Quick Reference

Constraint Type	Simple
Value Types	`ViaDefArrayValue`
Database Types	derived, internal only
Scope	design, foundry
Category	Routing

Value Type

`extendedValidRoutingVias` constraints have a `ViaDefArrayValue` that specifies the vias.

Related Topics

Routing Constraints

horizontalRouteGridOffset

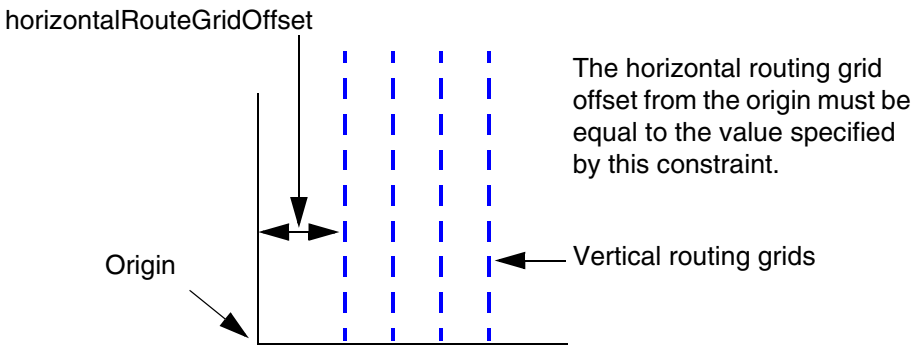
Sets the offset from the origin, in user units, for the start of the vertical routing grid. Along with the horizontalRouteGridPitch, this constraint defines the route grid points on the X axis for the layer.

horizontalRouteGridOffset Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`horizontalRouteGridOffset` constraints have a Value that represents the offset, in user units, from the original in the X direction for the layer.



Examples

In this example, the start for the vertical routing grid on `Metal1` is offset 1.5 from the origin in the X direction.

Format	Example
Tcl	`set_layer_constraint -constraint horizontalRouteGridOffset \` `-layer Metal1 -Value 1.5`

LEF	`Layer Metal1`
	`  TYPE ROUTING ;`
	`  OFFSET 1.5 ;`
	Note: This example sets the same value for both horizontal and vertical offsets. If the values differ, include both values using this syntax:
	`OFFSET` *`f_offset_x f_offset_y`* `;`

Virtuoso	`routingGrids (`
	`(horizontalOffset "Metal1" 1.5)`
	`)`

Related Topics

Routing Constraints

horizontalRouteGridPitch

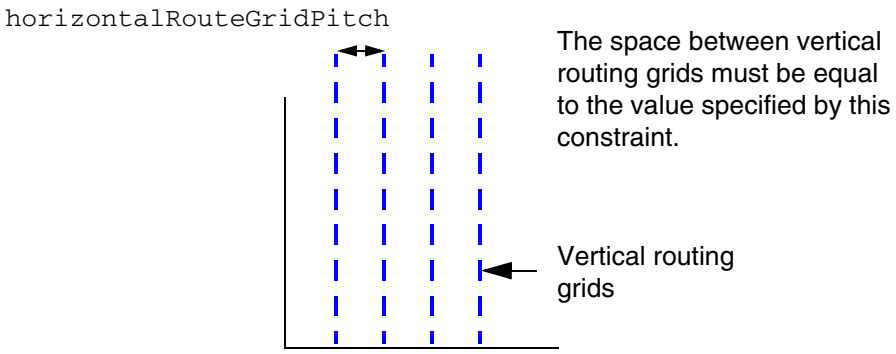
Sets spacing, edge-to-edge, in user units, between vertical routing grids. Along with the horizontalRouteGridOffset, this constraint defines the route grid points on the X axis for the layer.

horizontalRouteGridPitch Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`horizontalRouteGridPitch` constraints have a Value that represents the spacing, in user units, between vertical routing grids for the layer.



Examples

In this example, the spacing between vertical routing grids must be 0.6 user units on `Metal1`.

Format	Example
Tcl	`set_layer_constraint -constraint horizontalRouteGridPitch \ -layer Metal1 -Value 0.6`

Format	Example

LEF	`Layer Metal1`
	`  TYPE ROUTING ;`
	`  PITCH 0.6 ;`
	Note: This example sets the same value for both horizontal and vertical pitches. If the values differ, include both values with this syntax:
	`  PITCH` *`f_pitch_x f_pitch_y`* `;`

Virtuoso	`routingGrids (`
	`  (horizontalPitch "Metal1" 0.6)`
	`  )`

Related Topics

Routing Constraints

inlineViaPreferred

Specifies a preference for inline vias, with longer extensions aligned with the longer dimension of the cut bound, and will override the preferred extension direction of the upper layer.

This constraint is enabled by setting the `db.use_separate_pref_ext_dir` environment variable to `true`.

When this constraint is set, the create_derived_vias command will create vias with aligned extensions even if the `-align_extensions` argument is not set.

inlineViaPreferred Quick Reference

Constraint Type	Layer pair
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`inlineViaPreferred` constraints have a `BoolValue` that specifies whether inline vias are preferred for the given upper and lower layer pair.

Examples

The following example sets a preference for `Metal1`/`Metal2` vias to be inline.

```
set_layerpair_constraint -constraint inlineViaPreferred -layer1 Metal1 -layer2
Metal2 -BoolValue true
```

Related Topics

Routing Constraints

layerHeight

Specifies the height of the layer off the substrate.

layerHeight Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`layerHeight` constraints have a `Value` to specify the height, in user units, off the substrate.

Related Topics

Routing Constraints

layerThickness

Specifies the thickness of the layer.

layerThickness Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`layerThickness` constraints have a `Value` that specifies the thickness of the layer.

Related Topics

Routing Constraints

limitRoutingLayers

Limits routing to the layers specified by the constraint ANDed with the layers defined by the viaBarAdjacentSpacing constraint.

limitRoutingLayers Quick Reference

Constraint Type	Simple
Value Types	`LayerArrayValue`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, design, foundry
Category	Routing

Value Type

`limitRoutingLayers` constraints have a `LayerArrayValue` that specifies the layers that can be used for routing. Only layers in the list that are also specified by the `viaBarAdjacentSpacing` constraint can be used for routing.

Examples

This example restricts routing to layers `Metal1` and `Metal2`.

```
set_constraint -constraint limitRoutingLayers -LayerArrayValue {Metal1 Metal2}
```

Related Topics

Routing Constraints

limitViaThruLayers

Specifies layers that you cannot route on but can via through.

limitViaThruLayers Quick Reference

Constraint Type	Simple
Value Types	`LayerArrayValue`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, design, foundry
Category	Routing

Value Type

`limitViaThruLayers` constraints have a `LayerArrayValue` that lists the layers that you cannot route on but can via through.

Related Topics

Routing Constraints

maxPickupDistanceAllowed

Specifies the maximum distance, in user units, on a layer for a route connected to a pin on the same layer.

maxPickupDistanceAllowed Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design
Scope	design
Category	Routing

Value Type

`maxPickupDistanceAllowed` constraints have a Value that represents the maximum distance, in user units, on a layer for a route connected to a pin on the same layer.

Related Topics

Routing Constraints

maxRoutingDistanceAllowed

Specifies the maximum cumulative distance on the layer for all routes of a net.

maxRoutingDistanceAllowed Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`maxRoutingDistanceAllowed` constraints have a `Value` that specifies the cumulative distance on a given layer for all routes of a net.

Examples

This example sets the maximum routing distance on `Metal3` to `50` µm.

Format	Example
Tcl	`set_layer_constraint -constraint maxRoutingDistanceAllowed \` `  -layer Metal3 -Value 50`

Virtuoso	`interconnect(`
	`  (maxRoutingDistance "Metal3" 50)`
	`  …`
	`)`

Related Topics

Routing Constraints

maxTaperWindow

Specifies the distance away from a term at which the influence of the taper constraints stop and switches to the net's constraints. Between minTaperWindow and maxTaperWindow, the router is allowed to switch to the net's constraints at any time, usually cost-based, but once the switch away from taper constraints is made, it cannot be switched back.

maxTaperWindow Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	taper group for term or instTerm, inputTaper and/or outputTaper group for nets
Category	Routing

Value Type

`maxTaperWindow` constraints have a `Value` that represents the distance, in user units, away from a term at which the influence of the taper constraints stop and switches to the net's constraints.

Related Topics

Routing Constraints

mfgGrid

Specifies the `oaTech` physical layer attribute for the manufacturing grid of the layer.

mfgGrid Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	derived, internal only
Scope	design
Category	Routing

Value Type

The `mfgGrid` constraint has a `Value` that represents the manufacturing grid, in user units, for the layer.

Related Topics

Routing Constraints

minTaperWindow

Specifies the distance away from a terminal that *must* adhere to the taper rule constraints. Between `minTaperWindow` and `maxTaperWindow`, the router is permitted to switch to the net's constraints at any time, usually cost-based, but once the switch away from taper constraints is made, it cannot be switched back.

Note: Usually, tapering is used only when needed to connect a wide wire to a smaller terminal. Setting this constraint forces tapering within the `minTaperWindow` distance from the terminal.

minTaperWindow Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	taper group for term or instTerm, inputTaper and/or outputTaper group for nets
Category	Routing

Value Type

`minTaperWindow` constraints have a `Value` that represents the minimum distance from a terminal that must adhere to the taper rule constraints.

Examples

Format	Example
Tcl	`set_constraint -constraint minTaperWindow -Value f_distance`

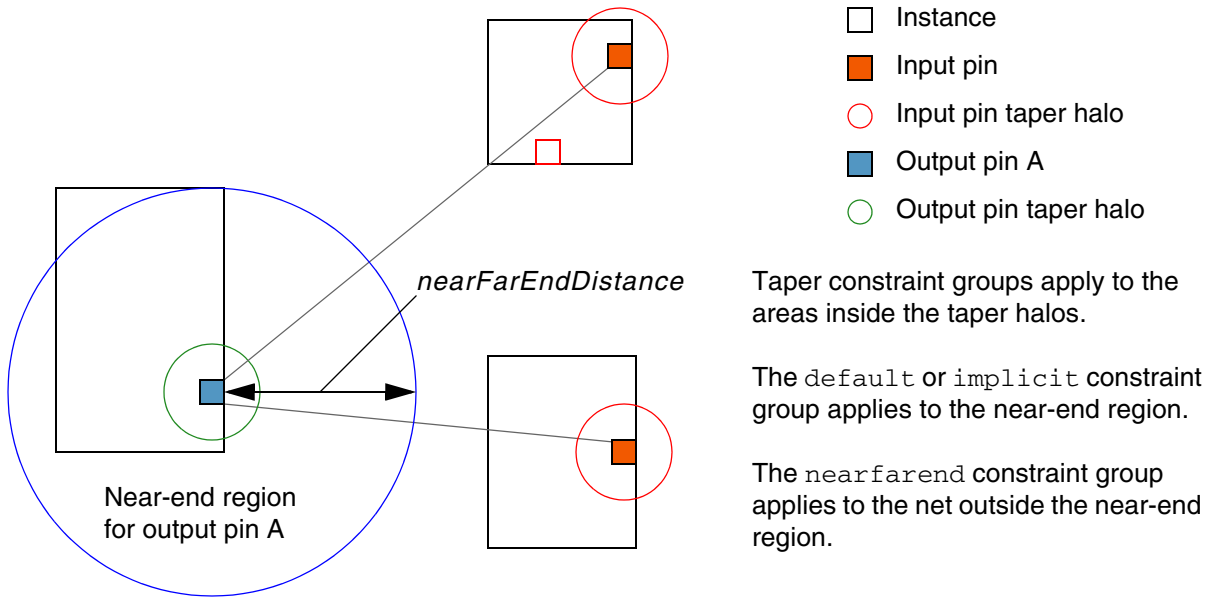
Related Topics

Routing Constraints

nearFarPercentage

Specifies the percentage of the entire net around the output pins that must be routed using the `default` or `implicit` constraint group, while the rest of the net is routed using the `nearfarend` constraint group. The constraint does not apply to power or ground nets, or to nets with no input pins. The constraint applies only to the output pins of the net. If the net has multiple output pins, then the rule applies to each of them.

The minimum distance between each output pin and its farthest input pin is calculated and the minimum of these is the radius of the net. The *`nearFarEndDistance`* is calculated by multiplying the `nearFarPercentage` by the radius, then dividing by 100. The `default` or `implicit` constraint group applies to the routing within *`nearFarEndDistance`* of each output pin (*near-end region*) and the `nearfarend` constraint group applies to the net outside the *near-end region*.



The `nearfarend` constraint group is recognized only by the global router and can contain the following constraints: `validRoutingLayers`, `validRoutingVias`, minWidth, minSpacing, and minNumCut.

nearFarPercentage Quick Reference

Constraint Type	Simple
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Routing
OpenAccess API	custom

Value Type

IntValue	Specifies the percentage of the entire net from the output pins that must be global routed using the `default` or `implicit` constraint group, with the rest of the net routed using the `nearfarend` constraint group. Valid values are 0 to 100. Tapers for the input and output pins override the `nearfarend` constraint group within their respective taper halos, except when the `nearFarPercentage` value is 0. For this case, the `nearfarend` constraint group will be used for the entire net. A value of 100 means that no calculation for the *near-end region* is needed because it will not apply for the net.

Examples

Creates a *near-end region* around the output pins of the nets in the selected set that is 55% of the minimum distance between the output pins and their farthest input pins, then global routes the nets. Within the *near-end regions*, the `default` or `implicit` constraint group for the net applies; outside that region, the `nearFarEnd` constraint group applies. If the `default` constraint group specifies layers with 4X width wires, and layers E2 and E3 specify 2X wires, then this example effectively routes the selected nets on 4X wires for 55% of the distance from the output pins and 2X wires for rest of the distance to the input pins.

```
create_constraint_group -name nearfar
set_constraint -constraint nearFarPercentage -IntValue 55 group nearfar
set_constraint -consraint validRoutingLayers -LayerArrayValue {E2 E3} -hardness
soft -group nearfar
set_constraint_group -set[get_selection_set] -nearfarend nearfar
deselect
global_route
```

Related Topics

Routing Constraints

oaDefault135RouteGridOffset

Specifies the default 135 degree (left diagonal) route grid offset for all layers and applies only to layers that do not have the 135RouteGridOffset constraint set.

oaDefault135RouteGridOffset Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Types

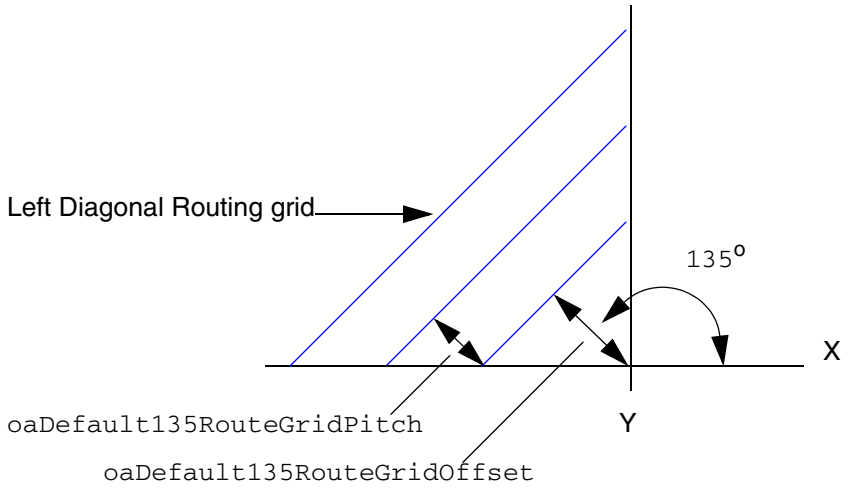
`oaDefault135RouteGridOffset` constraints have a `Value` that represents the default 135 degree route grid offset in user units.

Examples

Format	Syntax
Tcl	`set_constraint -constraint oaDefault135RouteGridOffset \` `-Value` *`f_offset`*

Virtuoso	`routingGrids(`
	`( leftDiagOffset` *`f_offset`* `)`
	`)`

Illustration of the example for `oaDefault135RouteGridOffset` **constraint**



Related Topics

Routing Constraints

oaDefault135RouteGridPitch

Specifies the default 135 degree (left diagonal) route grid pitch for all layers and applies only to layers that do not have the 135RouteGridPitch constraint set.

oaDefault135RouteGridPitch Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

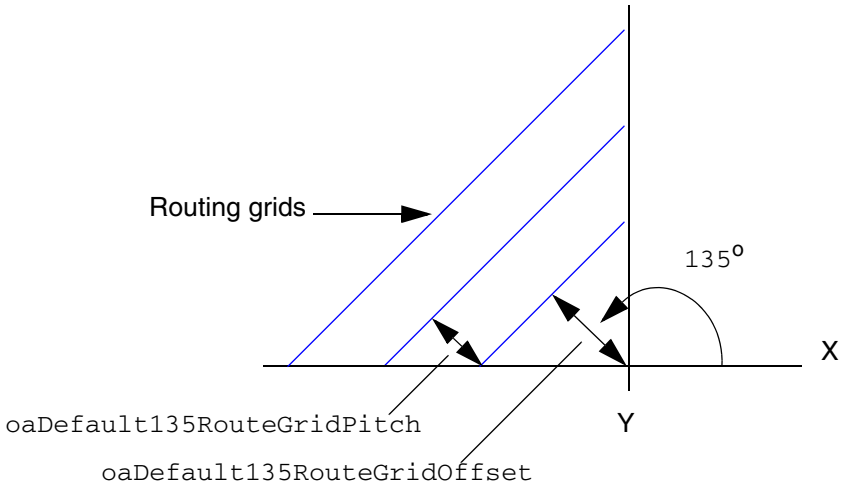
`oaDefault135RouteGridPitch` constraints have a `Value` that represents the default left diagonal route grid pitch in user units.

Examples

Format	Syntax
Tcl	`set_constraint -constraint oaDefault135RouteGridPitch \` `-Value` *`f_pitch`*

Virtuoso	`routingGrids(`
	`( leftDiagPitch` *`f_pitch`* `)`
	`)`

Illustration of the example for `oaDefault135RouteGridPitch` constraint



Related Topics

Routing Constraints

oaDefault45RouteGridOffset

Specifies the default 45 degree (right diagonal) route grid offset for all layers and applies only to layers that do not have the 45RouteGridOffset constraint set.

oaDefault45RouteGridOffset Quick Reference

Constraint Type	Simple
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

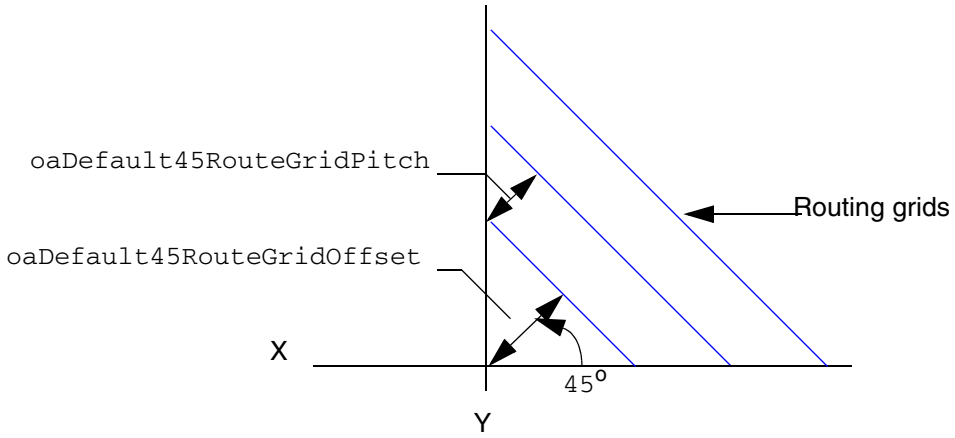
`oaDefault45RouteGridOffset` constraints have a Value that represents the default 45 degree route grid offset in user units.

Examples

Format	Syntax
Tcl	`set_constraint -constraint oaDefault45RouteGridOffset \` `-Value` *`f_offset`*

Virtuoso	`routingGrids(`
	`( rightDiagOffset` *`f_offset`* `)`
	`)`

Illustration of the example for `oaDefault135RouteGridPitch` constraint



Related Topics

Routing Constraints

oaDefault45RouteGridPitch

Specifies the default 45 degree (right diagonal) route grid pitch for all layers and applies only to layers that do not have the 45RouteGridPitch constraint set.

oaDefault45RouteGridPitch Quick Reference

Constraint Type	Simple
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Types

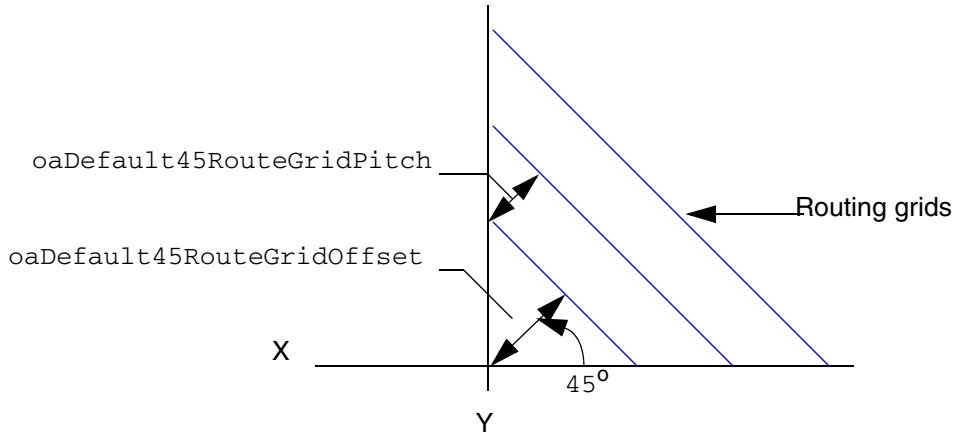
`oaDefault45RouteGridPitch` constraints have a Value that represents the default 45 route grid pitch in user units.

Examples

Format	Syntax
Tcl	`set_constraint -constraint oaDefault45RouteGridPitch \` `-Value` *`f_pitch`*

Virtuoso	`routingGrids(`
	`( rightDiagPitch` *`f_pitch`* `)`
	`)`

Illustration of the example for `oaDefault135RouteGridPitch` **constraint**



Related Topics

Routing Constraints

oaPreferredRoutingDirection

The `oaPreferredRoutingDirection` constraint is specific to the default constraint group of an `oaAreaBoundary`. The preferred routing direction outside the oaAreaBoundary is specified in the technology database. The preferred routing direction can be one of the orthogonal or one of the diagonal routing directions.

oaPreferredRoutingDirection Quick Reference

Constraint Type	Layer
Value Types	`IntValue`, `StringAsIntValue`
Database Types	Design, Technology
Scope	area boundary, design, foundry
Category	Routing

Value Types

`oaPreferredRoutingDirection` constraints have an `IntValue` or a `StringAsIntValue` that is input as a string value and stored as an integer value as shown in the following table.

oaPreferredRoutingDirection Values

String	Integer Value	Preferred Routing Direction:
`notApplicablePrefRoutingDir`	0	Preferred direction cannot be determined
`nonePrefRoutingDir`	1	All directions allowed
`horzPrefRoutingDir`	2	
`vertPrefRoutingDir`	3	

Examples

In this example, applying the `myRoutingLayerOverrides` constraint group to a bounded area the following routing direction will be set within that area:

- `Metal1` to `vertical`
- `Metal2` to `horizontal`
- `Metal3` to none, thereby allowing it to take any routing direction

Format	Example

Tcl	`set_layer_constraint -constraint oaPreferredRoutingDirection \`
	`  -layer Metal1 -StringAsIntValue vertPrefRoutingDir \`
	`  -group myRoutingLayerOverrides`
	`set_layer_constraint -constraint oaPreferredRoutingDirection \`
	`  -layer Metal2 -StringAsIntValue horzPrefRoutingDir \`
	`  -group myRoutingLayerOverrides`
	`set_layer_constraint -constraint oaPreferredRoutingDirection \`
	`  -layer Metal3 -StringAsIntValue nonePrefRoutingDir \`
	`  -group myRoutingLayerOverrides`

Virtuoso	`constraintGroups(`
	`  ("myRoutingLayerOverrides"`
	`  routingDirections(`
	`  ("Metal1" "vertical")`
	`  ("Metal2" "horizontal")`
	`  ("Metal3" "none")`
	`  )`
	`  )`
	`)`

Related Topics

Routing Constraints

oaTaperHalo

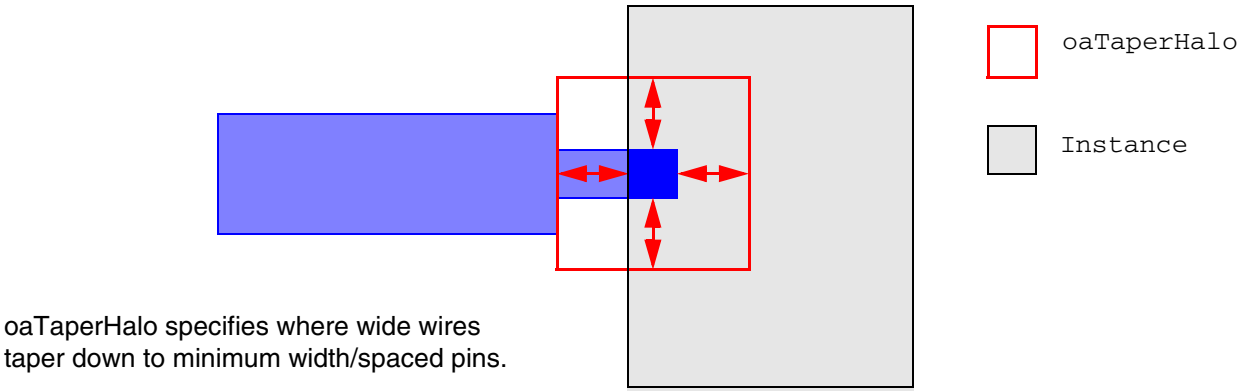
Establishes a window, or halo, around a pin in which wide wires are to be tapered in accordance with the spacing and width constraints in taper constraint group.

oaTaperHalo Quick Reference

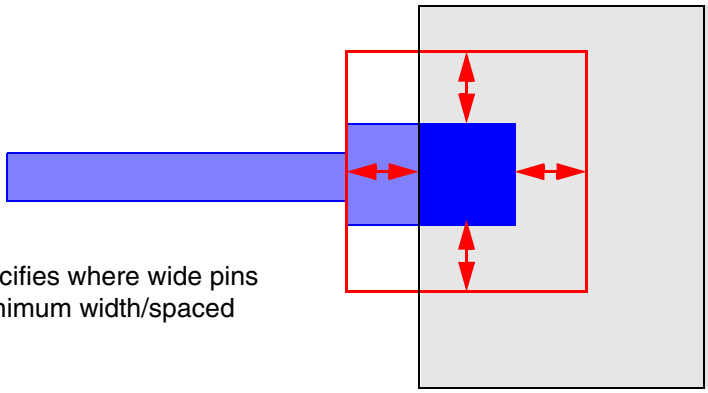
Constraint Type	Simple
Value Types	Value
Database Types	Design, Technology
Scope	taper group for term or instTerm, inputTaper and/or outputTaper group for nets
Category	Routing

Value Type

`oaTaperHalo` constraints have a `Value` that is the offset, in user units, around the pin shapes that represents the taper window.



oaTaperHalo specifies where wide wires taper down to minimum width/spaced pins.



oaTaperHalo specifies where wide pins taper down to minimum width/spaced wires.

Examples

This example establishes a 1.5 offset, in user units, for the taper halo used to taper wires to pins.

Format	Example
Tcl	`set_constraint -constraint oaTaperHalo -Value 1.5`
Virtuoso	`spacings( taperHalo 1.5) )`

Related Topics

Routing Constraints

offset

Specifies the offset to the routing grid for centerlines and endpoints of route segments, and the origins of vias to coincide with.

offset Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	design
Scope	default constraint group for nets and routes
Category	Routing

Value Type

`offset` constraints have a `Value` that represents the offset, in user units.

Related Topics

Routing Constraints

orthogonalSnappingLayer

Specifies the stopping points to align end-of-line edges on *layer1* with the routing grid on *layer2*, based on the widths of the routing tracks. The routing grid on *layer2* can be specified using a number of ways, including track patterns, snapPatternDefs, and widthSpacingSnapPatternDefs. Both layers must have orthogonal direction.

orthogonalSnappingLayer Quick Reference

Constraint Type	Layer-pair (non-symmetric)
Value Types	RangeArray1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

RangeArray1DTblValue	Specifies the stopping points to align end-of-line edges on *layer1* with the routing grid on *layer2*, based on the widths of the routing tracks.

Required Parameters

None

Optional Parameters

`upperLineEndStoppingPoints`

Specifies in user units the stopping-point distances for the right and upper edges of the shapes on *layer1*; the constraint value specifies the stopping point distances for the left and lower edges of the shapes on *layer1*.

If this parameter is not specified, the constraint value applies to all edges, lower and left and upper and right.

Type: RangeArray1DTblValue

Examples

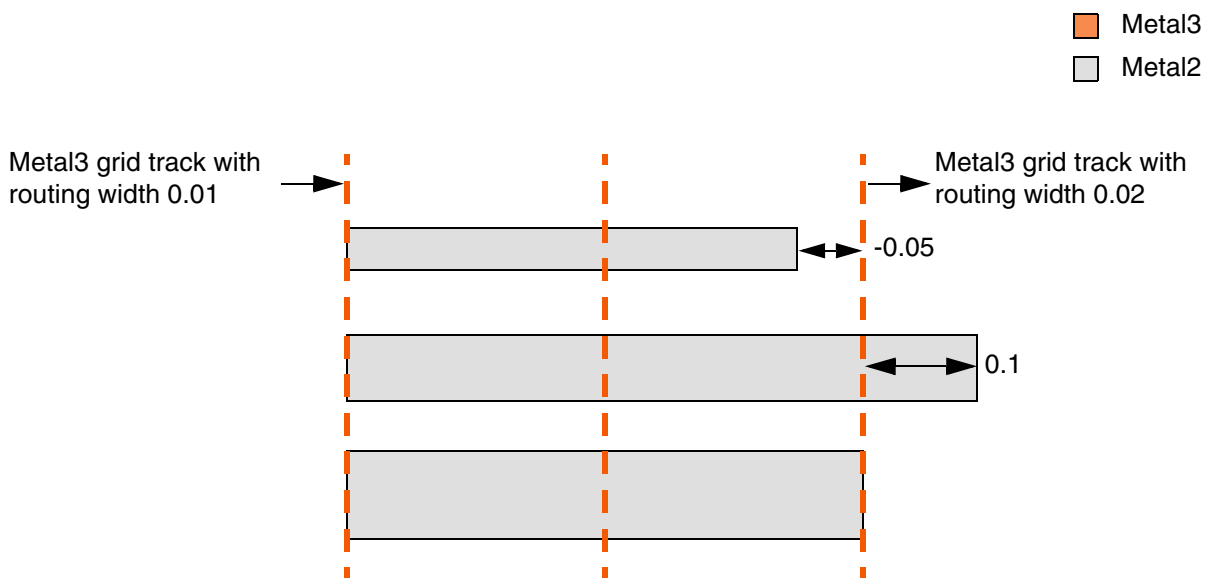
```
set_layerpair_constraint -layer1 Metal2 -layer2 Metal3 \
  -constraint orthogonalSnappingLayer -row_name width \
  -RangeArray1DTblValue {0.0    1   0 \
                         0.02   3   -0.05  0  0.1}
```

Specifies that the end-of-line edges of `Metal2` shapes must stop as follows:

- At a distance equal to 0 from the centerline of the track if the width of a routing track on `Metal3` is greater than or equal to 0.0.
- At a distance equal to -0.05, 0, or 0.1 from the centerline of the track if the width of a routing track on `Metal3` is greater than or equal to 0.02.

The table is indexed by track width. Each row contains a track width, the count of `RangeArrayValue` for the given track width, and the `RangeArrayValue` list that specifies the allowed stopping-point distances.

Illustration for orthogonalSnappingLayer



PASS. The top Metal2 wire uses the -0.05 stopping point. The middle Metal2 wire uses the 0.1 stopping point. The bottom wire uses the 0 stopping point. As a result, all three Metal2 shapes pass.

Related Topics

Routing Constraints

check_layerpair_space

planarTapAllowed

Specifies whether routes can be connected to another leg of the net in the same layer at a Steiner point.

planarTapAllowed Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design
Category	Routing

Value Type

`planarTapAllowed` constraints have a `BoolValue`.

Related Topics

Routing Constraints

preferredExtensionDirection

Specifies the preferred direction for via extensions on the given layer. A via with extensions aligned to the `preferredExtensionDirection` of the lower and upper metal layers will have the lowest cost and will be preferred for routing over another via that has the same number of cuts and the same extension dimensions but violates this constraint.

This constraint is enabled by setting the `db.use_separate_pref_ext_dir` environment variable to `true`. By default, when `db.use_separate_pref_ext_dir` is set `false`, the preferred direction for the longer via extent is given by `oaPreferredRoutingDirection`.

When this constraint is set, the create_derived_vias command will create preferred/non-preferred via extensions based on this constraint.

preferredExtensionDirection Quick Reference

Constraint Type	Layer
Value Types	`StringAsIntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`preferredExtensionDirection` constraints have a `StringAsIntValue` that is input as a string value and stored as an integer value as shown in the following table.

preferredExtensionDirection Values

String	Integer Value	Preferred Extension Direction:
`horzPrefRoutingDir`	2	
`vertPrefRoutingDir`	3	

Examples

In this example, the preferred extension direction for `Metal3` is set to vertical.

```
setvar db.use_separate_pref_ext_dir true
set_layer_constraint -constraint preferredExtensionDirection -layer Metal3
-StringAsIntValue vertPrefRoutingDir
```

Related Topics

Routing Constraints

preferredViaOrigin

Specifies the via origin preference as offset or non-offset vias.

This constraint is enabled by setting the `db.use_separate_pref_ext_dir` environment variable to `true`.

preferredViaOrigin Quick Reference

Constraint Type	Layer pair
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`preferredViaOrigin` constraints have an `IntValue` as shown in the following table.

preferredViaOrigin Values

Integer Value	Description
0	No preference
1	Prefer via with origin at the center of all cuts
2	Prefer offset vias

Related Topics

Routing Constraints

routeOnGrid

Specifies whether routing must be on-grid or can be off-grid.

routeOnGrid Quick Reference

Constraint Type	Simple
Value Type	`BoolValue`
Database Types	Design, Technology
Scope	design
Category	Routing

Value Type

`routeOnGrid` constraints have a `BoolValue`.

Related Topics

Routing Constraints

taperToFirstVia

Specifies whether the taper window is from the instance pins to the first layer change (first via) or bend. When this constraint is not set or is set to `false`, the taper window is determined by the settings of `oaTaperHalo`, `maxTaperWindow`, and `minTaperWindow`, as specified in *Using Tapers*.

taperToFirstVia Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing
OpenAccess API	custom

Value Type

`taperToFirstVia` constraints have a `BoolValue`.

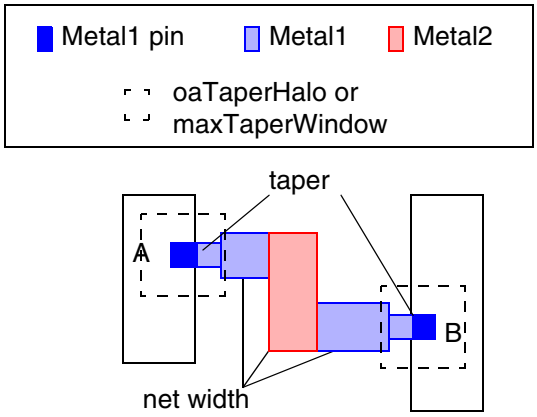
`true`	The taper window is from the instance pins to the first layer change (first via) or bend.
`false` or not set	The taper window is determined by the settings of `oaTaperHalo`, `maxTaperWindow`, and `minTaperWindow`.

Examples

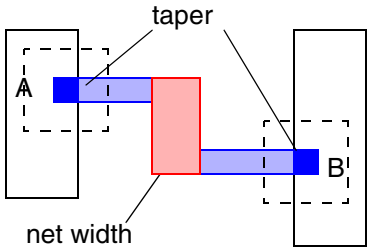
This example sets tapering from the instance pins to the first layer change or bend for objects assigned to the taper1 taper spec.

```
set_constraint -constraint taperToFirstVia -BoolValue true -group taper1
```

Examples for taperToFirstVia



a) When taperToFirstVia is not set or is set to `false`, tapering occurs within the taper window when needed, as in this case where the net width is wider than the pin width.



b) When taperToFirstVia is set to `true`, tapering is from the pin to the first layer change (first via) or bend, when needed.

Related Topics

Routing Constraints

TJunctionAllowed

Specifies whether the point-to-point router can tap into an existing route segment to complete connections.

TJunctionAllowed Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`TJunctionAllowed` constraints have a `BoolValue`. When `TJunctionAllowed` is set to `false` in a constraint group assigned to a route segment, the point-to-point router cannot tap into the route segment. When `TJunctionAllowed` is set to `true` or is not set in a constraint group assigned to a route segment, the point-to-point router can tap into the route segment.

Related Topics

Routing Constraints

validRoutingLayers

Specifies the layers that can be used when routing.

validRoutingLayers Quick Reference

Constraint Type	Simple
Value Types	LayerArrayValue
Database Types	Design, Technology
Scope	net, route, Term, instTerm, net group, design, foundry
Category	Routing

Value Type

`validRoutingLayers` constraints have a LayerArrayValue that is a list of layers that can be used when routing.

Examples

This example sets the valid routing layers to `Metal1`, `Metal2`, and `Metal3`.

Format	Example
Tcl	`set_constraint -constraint validRoutingLayers \` `-LayerArrayValue {Metal1 Metal2 Metal3}`

Virtuoso	`interconnect(`
	`(validLayers ("Metal1" "Metal2" "Metal3" ))`
	`)`

Related Topics

Routing Constraints

validRoutingVias

Specifies the list of vias that can be used when routing.

validRoutingVias Quick Reference

Constraint Type	Simple
Value Types	ViaDefArrayValue
Database Types	Design, Technology
Scope	net, route, Term, instTerm, net group, design, foundry
Category	Routing

Value Type

`validRoutingVias` constraints have a ViaDefArrayValue that represents the list of vias that can be used when routing.

Examples

This example sets the valid routing vias to `via1` and `via2`.

Format	Example
Tcl	`set_constraint -constraint validRoutingVias \` `-ViaDefArrayValue {via1 via2}`

Virtuoso	`interconnect(`
	`(validVias ("via1" "via2"))`
	`)`

Related Topics

Routing Constraints

validWireEditorVias

Specifies the vias that can be used during wire editing. If this constraint is not set, `validRoutingVias` is used.

validWireEditorVias Quick Reference

Constraint Type	Simple
Value Types	`ViaDefArrayValue`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, net group, design, foundry
Category	Routing
OpenAccess API	custom

Value Type

`validWireEditorVias` constraints have a `ViaDefArrayValue` that represents the list of vias that can be used during wire editing.

Examples

This example sets the valid wire editor vias for the `LEFDefaultRouteSpec` group to `via12` and `via23`.

```
set_constraint -constraint validWireEditorVias -group LEFDefaultRouteSpec \
  -hardness hard -ViaDefArrayValue {via12 via23}
```

Related Topics

Routing Constraints

verticalRouteGridOffset

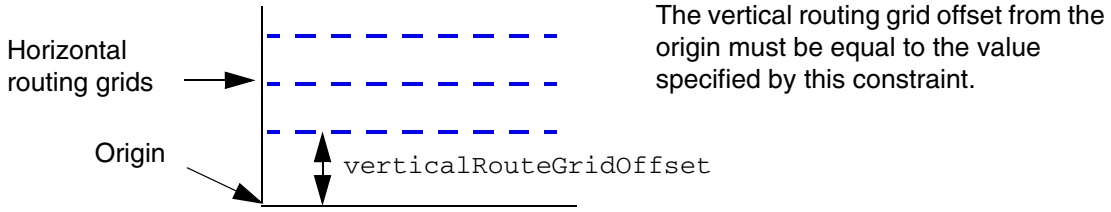
Sets the offset from the origin, in user units, for the start of the vertical routing grid. Along with the horizontalRouteGridPitch, this constraint defines the route grid points on the X axis for the layer.

verticalRouteGridOffset Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Type

`verticalRouteGridOffset` constraints have a `Value` that represents the offset, in user units, from the original in the Y direction for the layer.



Examples

In this example, the start for the vertical routing grid on `Metal1` is offset 1.5 from the origin in the X direction.

Format	Example
Tcl	`set_layer_constraint -constraint verticalRouteGridOffset \ -layer Metal1 -Value 1.5`

Format	Example

LEF	`Layer Metal1`
	`  TYPE ROUTING ;`
	`  OFFSET 1.5 ;`
	Note: This example sets the same value for both vertical and vertical offsets. If the values differ, include both values using this syntax:
	`  OFFSET` *`f_offset_x f_offset_y`* `;`

Virtuoso	`routingGrids (`
	`  (verticalOffset "Metal1" 1.5)`
	`  )`

Related Topics

Routing Constraints

verticalRouteGridPitch

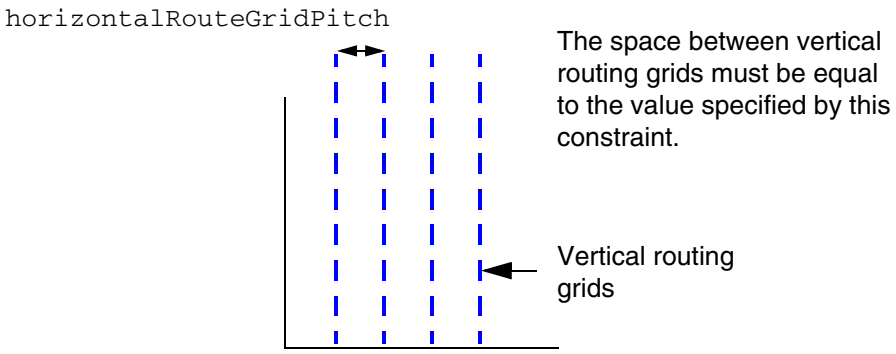
Sets spacing, edge-to-edge, in user units, between vertical routing grids. Along with the horizontalRouteGridOffset, this constraint defines the route grid points on the X axis for the layer.

verticalRouteGridPitch Quick Reference

Constraint Type	Simple
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Routing

Value Types

`verticalRouteGridPitch` constraints have a `Value` that represents the spacing, in user units, between vertical routing grids for the layer.



Examples

In this example, the spacing between vertical routing grids must be 0.6 user units on `Metal1`.

Format	Example
Tcl	`set_layer_constraint -constraint verticalRouteGridPitch \` `-layer Metal1 -Value 0.6`

Format	Example

LEF	`Layer Metal1`
	`  TYPE ROUTING ;`
	`  PITCH 0.6 ;`
	
	Note: This example sets the same value for both vertical and vertical pitches. If the values differ, include both values with this syntax:
	
	`  PITCH` *`f_pitch_x f_pitch_y`* `;`

Virtuoso	`routingGrids (`
	`  (verticalPitch "Metal1" 0.6)`
	`  )`

Related Topics

Routing Constraints

viaTapAllowed

Specifies whether a route can be connected to another leg of the net with a via.

viaTapAllowed Quick Reference

Constraint Type	Simple
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design
Category	Routing

Value Type

The `viaTapAllowed` constraint has a `BoolValue` that specifies whether a route can be connected to another leg of the net with a via.

Related Topics

Routing Constraints

wireExtent

Specifies the general wire extension.

wireExtent Quick Reference

Constraint Type	Layer
Value Types	`Value`
Category	Routing

Value Type

The `wireExtent` constraint has a `Value` that specifies the general wire extension, in user units.

Related Topics

Routing Constraints

wrongWayOK

Specifies whether the router can create wrong-way routes on a layer. By default, wrong-way routing is permitted. When `wrongWayOk` is set to `false`, the router will not create wrong-way routes on the layer.

Note: Setting `wrongWayOk` to `false` does not prevent `fix_errors` from adding a wrong-way segment to fix an error.

wrongWayOK Quick Reference

Constraint Type	Layer
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design
Category	Routing

Value Type

`wrongWayOK` constraints have a `BoolValue`.

Examples

This example prevents the router from creating wrong-way routes on layer `Metal6`.

```
set_layer_constraint -layer Metal6 -constraint wrongWayOK -BoolValue false
```

Related Topics

Routing Constraints

16

Spacing Constraints

This topic lists the spacing constraints:

checkSpaceAsMaxXY	endOfLineKeepout	forbiddenEdgePitchRange
forbiddenProximitySpacing	maxTapSpacing	mergeSpaceAllowed
minBoundaryInteriorHalo	minCenterToCenterSpacing	minClusterSpacing
minDiagonalSpacing	minDiffIslandParallelViaSpacing	minDiffPotentialSpacing
minEdgeLengthSpacing	minEnclosedSpacing	minEndOfLineCutSpacing
minEndOfLineExtensionSpacing	minEndOfLinePerpSpacing	minEndOfLineSpacing
minEndOfStubSpacing	minExtensionSpacing	minInnerVertexSpacing
minJointCornerSpacing	minNotchSpanSpacing	minOppositeSpanSpacing
minOuterVertexSpacing	minParallelSpanSpacing	minProtrudedProximitySpacing
minProtrusionSpacing	minProximitySpacing	minSameNetSpacing
minSpacing	minVoltageSpacing	oaAllowedSpacingRange
oaMinEndOfNotchSpacing	oaMinNotchSpacing	shapeRequiredBetweenSpacing
trimMinSpacing		

Related Topics

Tcl Constraint Commands

Manage Constraints

checkSpaceAsMaxXY

This constraint is obsolete. To specify horizontal or vertical measurements, use the `oaSpacingDirection` parameter for constraints that support directional measurements.

endOfLineKeepout

In some processes, the spacing from the end of a line to other shapes on the same layer is specified by a simple keep-out region around the end of the line. Depending on the process, if a corner of another shape or a corner or an edge intersect the keep-out area, it may cause a violation. However, in some processes, if the corner or edge that intersects the keep-out area is part of the same metal shape, it is not considered as a violation. This rule applies to wires where ends of lines are less than a given width.

endOfLineKeepout Quick Reference

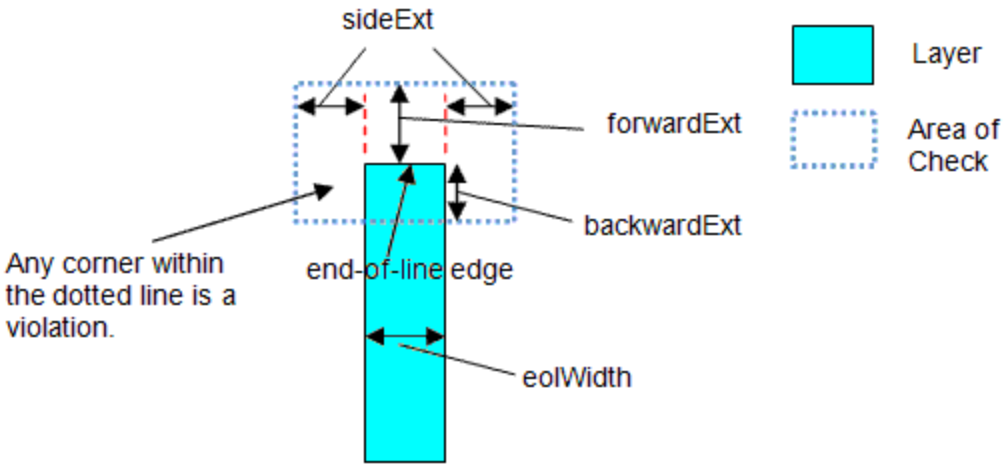
Constraint Type	Layer (non-symmetric)
Value Types	ValueArrayValue
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing
Group Operators	AND, OR

Value Type

ValueArrayValue

Specifies that the value array should contain the following three values in the following order:

- backwardExt defines the part of the keep-out area that extends back from the end-of-line edge
- sideExt defines the lateral dimensions of the keep-out area that extend from the sides of the end-of-line edge
- forwardExt specifies the keep-out region that is forward from the end-of-line edge



Required Parameters

`width`
: Specifies that the constraint applies for all lines with `eolWidth` less than the value of `width`.

  Type: `Value`

Optional Parameters

`cornerOnly`
: Specifies that only a corner is considered as a violation. If this parameter is not present or is set to `false` (the default), it is considered a violation if either an edge or a corner of another shape intersects the keep-out region.

  Type: `BoolValue`

`exceptConnectivityType`
: When set to 2 (contiguousShapesConnectivityType), the constraint does not apply to the edges and corners of the same metal shape. If this parameter is not specified, the constraint applies to all shapes.

  Type: `StringAsIntValue`
  0 anyConnectivityType
  1 sameNetConnectivityType
  2 contiguousShapesConnectivityType
  3 directShapesConnectivityType
  4 sameViaConnectivityType

`maxParallelRunLength`

Specifies that the constraint applies if the parallel run length between the end-of-line shape and the other shape is less than this value. (Also supports negative parallel run length.)

Type: `Value`

`exceptFrom`

Specifies that the rule applies only if the neighbor wire is not coming from either the front edge (when set to 1) or the back edge (when set to 2) of the keepout region.

Type: `IntValue`
1 exceptFromFrontEdge
2 exceptFromBackEdge

`twoSides`

Specifies that the rule applies only if the shape under consideration has different mask wires on either side.

Type: `BoolValue`

`colorMask`

Specifies that the constraint applies only to trigger wires on the specified mask

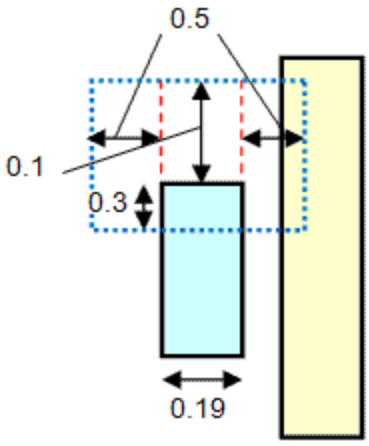
Type: `IntValue`
0 any
1 mask1
2 mask2
3 mask3

Examples

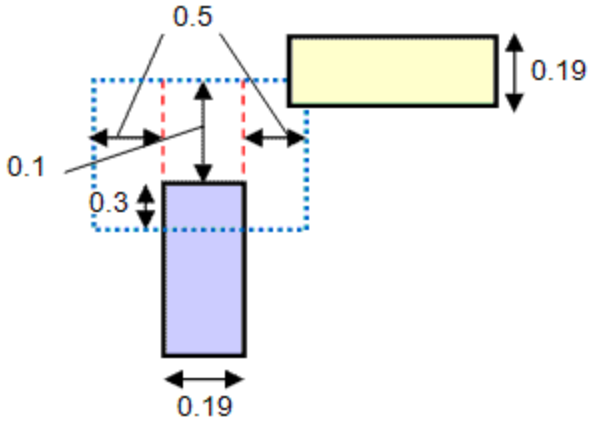
```
set_constraint_parameter -name width -Value 0.2
set_constraint_parameter -name cornerOnly -BoolValue false
set_layer_constraint -group foundry -layer Metal2 \
  -constraint endOfLineKeepout -hardness hard -ValueArrayValue (0.3 0.5 0.1)
```

Specifies a simple keep-out region around the end of the line under the following conditions:

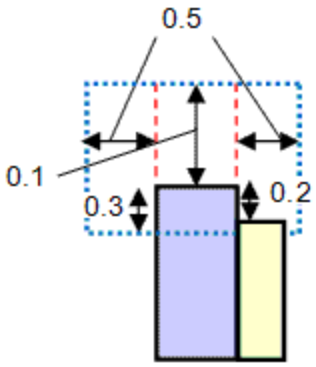
- All lines have the end-of-line width less than 0.2
- `cornerOnly` is set to `false`
- Constraint applies to all shapes.



a) Constraint violated as side neighbor intrudes into keep-out region. If cornerOnly was true, this would not be a violation, as only an edge intersects the keep-out area.



b) Constraint violated. Any neighbor in the dotted region is a violation.



c) Constraint violated. Any neighbors, including same-metal, would be a violation.

width = 0.2;
backwardExtension = 0.3;
sideExtension = 0.5;
forwardExtension = 0.1;
cornerOnly = false;
exceptSameMetal not defined.

Related Topics

Spacing Constraints

check_space

endOfLineKeepout

forbiddenEdgePitchRange

Specifies a restriction on the distance between the leftmost (or rightmost) edges of two shapes on a layer when there is a wire from a different metal island between those two shapes. The constraint also applies to two topmost or bottommost edges and only to wires of less than a given width. You can also optionally specify a parallel run length between the two edges and a wire in between and a minimum distance for neighboring wires on both sides.

forbiddenEdgePitchRange Quick Reference

Constraint Type	Layer
Value Types	RangeValue
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

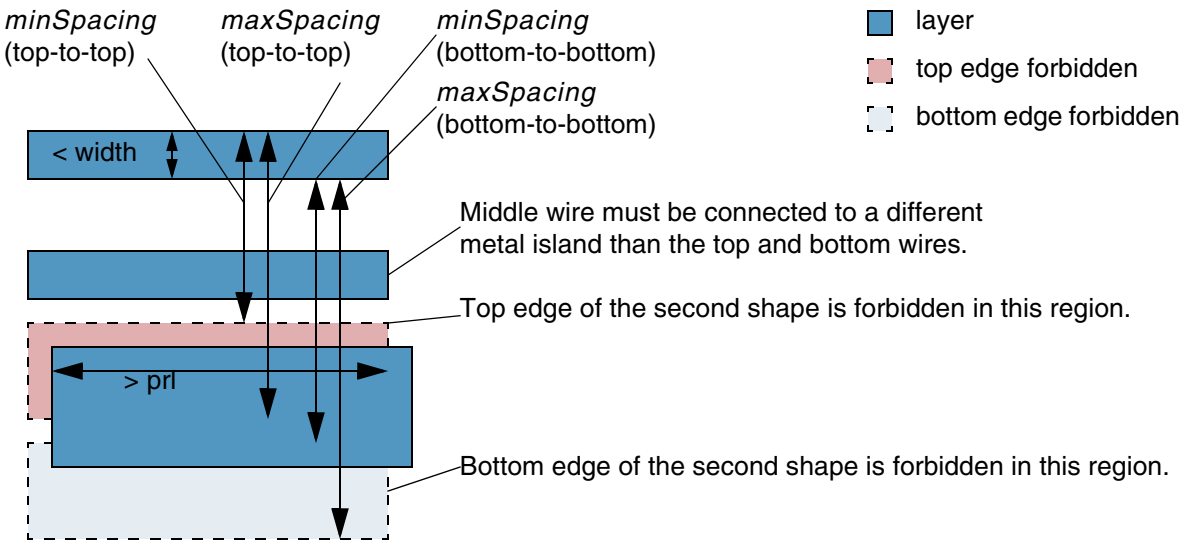
Value Type

RangeValue

Specifies the spacing, in user units, between the two shapes that is considered illegal. Any spacing that falls within the range is considered an illegal spacing.

For example, if the range is given as "[*minSpacing maxSpacing*]", then any spacing where *minSpacing* <= spacing <= *maxSpacing* is considered an illegal spacing, as shown in the following figure.

forbiddenEdgePitchRange Constraint



Required Parameter

`width`	Specifies that the constraint applies only for a wire with width less than this value in user units. Type: `Value`

Optional Parameters

`parallelRunLength`	Specifies that the constraint applies only if the common parallel run length between the two edges and the wire in between is greater than this value in user units. Type: `Value`
`parallelWithin`	Specifies that the constraint applies only if the wire has neighboring wires on both sides at a distance less than this value in user units. Type: `Value`

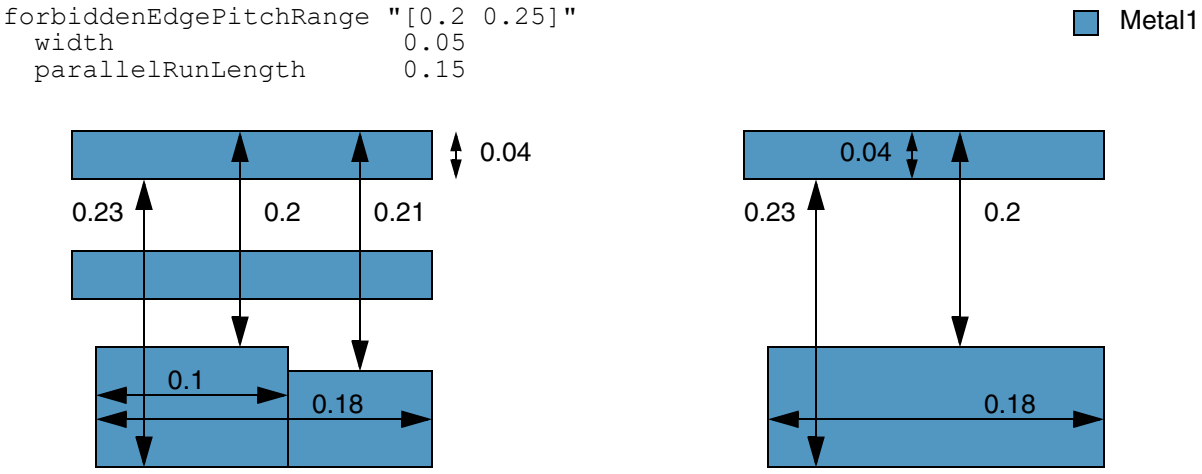
Examples

```
set_constraint_parameter -name width -Value 0.05
set_constraint_parameter -name parallelRunLength -Value 0.15
set_layer_constraint -constraint forbiddenEdgePitchRange -layer Metal1 \
  -RangeValue {"[0.2 0.25]"}
```

For a `Metal1` shape of width less than 0.05, the distance between it and another `Metal1` shape with a wire on a different metal island and a parallel run length greater than 0.15, the top-to-top edge spacing and bottom-to-bottom edge spacing must be greater than or equal to 0.02 and less than or equal to 0.25. The same applies for leftmost and rightmost edge-to-edge spacing on `Metal1`.

Illustration for forbiddenEdgePitchRange

```
forbiddenEdgePitchRange "[0.2 0.25]"
  width                  0.05
  parallelRunLength      0.15
```



a) FAIL. There is an unconnected wire between the top and bottom wires. The constraint does not apply for the top edges with parallel run lengths of 0.1 and 0.08 (< 0.15 `parallelRunLength`). The bottom edge has a parallel run length of 0.18 (> 0.15), so the constraint applies, causing a violation because the bottom-to-bottom edge spacing is 0.23 which is in the forbidden bottom edge region (>= 0.2 and <= 0.25).

b) Constraint does not apply because there is no wire between the top and bottom wires.

c) Constraint does not apply because the middle wire is on the same net island as the top wire.

Related Topics

Spacing Constraints

check_space

forbiddenProximitySpacing

Specifies the spacing between two wire shapes on a given layer. This constraint is violated if there are two wire shapes on a given layer, which are spaced apart by a distance greater than or equal to a given minimum spacing and less than or equal to a given maximum spacing.

In addition, both wires are within a given distance from a wide wire shape on the same layer whose width is greater than or equal to a given minimum width and has a parallel run length with the two wires greater than the specified length.

This constraint was formerly known as `forbiddenSpacingRange`.

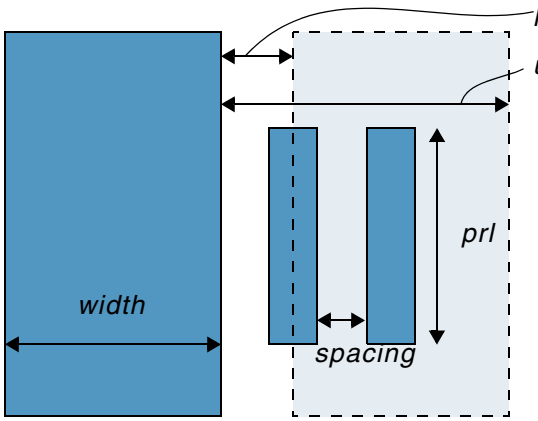
forbiddenProximitySpacing Quick Reference

Constraint Type	Layer
Value Types	`RangeValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`RangeValue`	Specifies the spacing, in user units, between the two shapes that is considered illegal. Any spacing that falls within the range is forbidden. For example, if RangeValue is `"[`*minSpacing maxSpacing*`]"`, then any spacing where *minSpacing* <= spacing <= *maxSpacing* is illegal. For this example, spacing between the two shapes must be less than *minSpacing* or greater than *maxSpacing*.

Figure 16-1 forbiddenProximitySpacing Constraint



The spacing between the two wires with parallel run length > `parallelRunLength` must not be within a region given by the constraint value if the gap between the wires is within a given `withinRange` of a wire with width >= `width`. In this example, `withinRange` is the area between *lowerWithinRange* and *upperWithinRange* from the wide wire.

Required Parameter

`width`

Specifies that the constraint applies only for a wire with width less than this value in user units.

Type: `Value`

Optional Parameters

`parallelRunLength`

Specifies that the constraint applies only if the parallel run length (*prl*) between the two edges is greater than this value, in user units.

Type: `Value`

`withinRange`

Specifies that the constraint applies to the spacing between two wires if the distance from the wide wire to the gap between the two wires lies in the range specified by this range value, in user units. For example, if RangeValue is `"[`*lowerWithinRange upperWithinRange*`]"`, then the constraint applies if the distance from the gap to the wide wire is *lowerWithinRange<=distance<=upperWithinRange.*

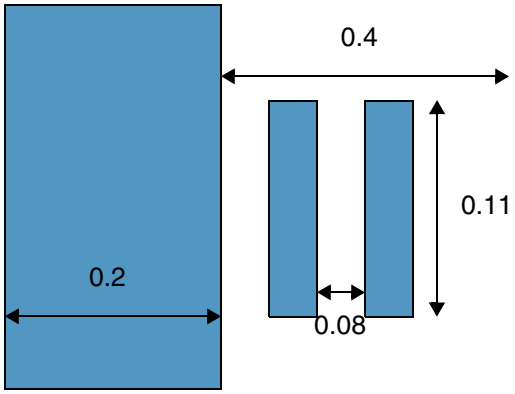
Type: `RangeValue`

Example

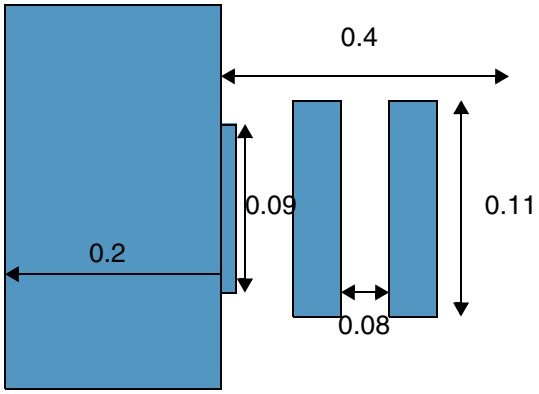
```
set_constraint_parameter -name width -Value 0.2
set_constraint_parameter -name parallelRunLength -Value 0.1
set_constraint_parameter -name withinRange -RangeValue {"<=0.4"}
set_layer_constraint -constraint forbiddenProximitySpacing \
  -layer m1 -RangeValue {"[0.07 0.09]"}
```

Specifies that two wires that are within 0.4 user units of a wire with width >= 0.2 user units and have a parallel run length > 0.1, must be < 0.07 or > 0.09 user units from each other.

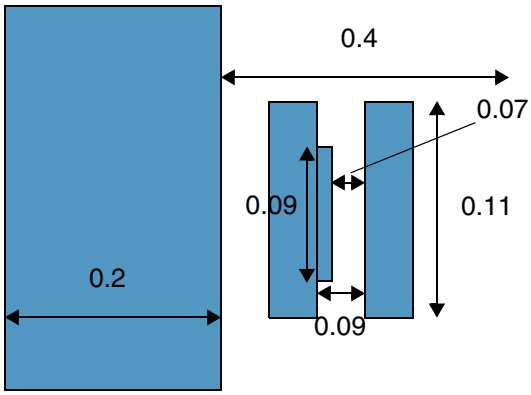
Illustration for forbiddenProximitySpacing



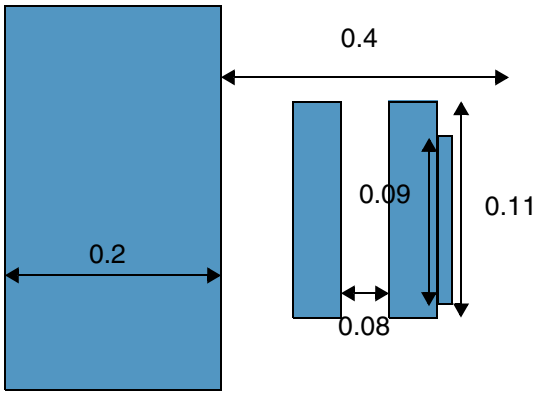
a) FAIL. The two neighbor wires have a spacing of 0.08, which is within the forbidden spacing of [0.07 0.09] and they have a parallel run length of 0.11 > 0.1 with a wide wire of width 0.2.



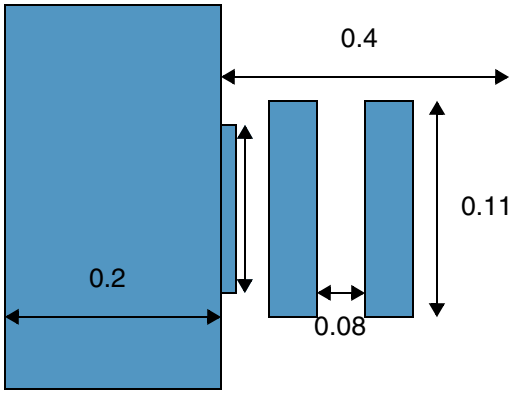
b) Constraint does not apply. The part of the wide wire with a width of 0.2 has a parallel run length of only 0.09 with the small neighbor wires.



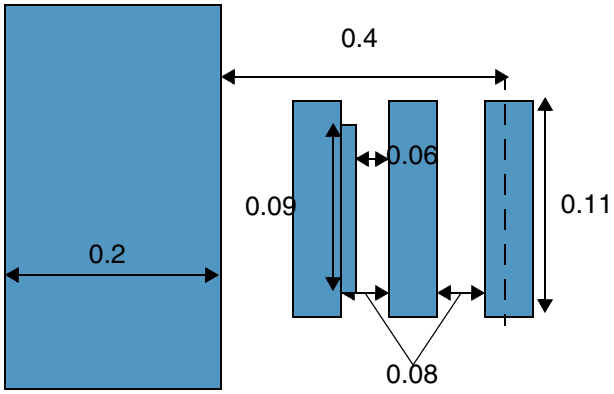
c) FAIL. The constraint applies because the wide wire is 0.2 (>= 0.2 `width`) and the parallel run length is measured over all the sections of the narrow wire and is 0.11 (>0.1). The 0.07 spacing at the jog and 0.09 for the rest of the wire are in the forbidden spacing range.



d) FAIL. A jog on the other side does not exempt the rule.



e) FAIL. Similar to a). Parallel run length > 0.1. The small jog on the wide wire is irrelevant.



f) FAIL. The jog with spacing of 0.06 on the first narrow wire would exempt the rule to the second wire, but the third wire is within the `withinRange` of the wide wire and its 0.08 spacing to the second wire is forbidden.

Related Topics

Spacing Constraints

check_space

maxTapSpacing

Specifies the maximum spacing required between two well ties.

maxTapSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`maxTapSpacing` constraints have a `Value` that represents the maximum spacing required between two well ties.

Related Topics

Spacing Constraints

check_space

mergeSpaceAllowed

Sets the threshold at which geometries on the same layer that do not meet the required minimum spacing can be merged into one shape. If the distance between the two geometries is smaller than the specified value, you can merge (fill the space) between the geometries.

mergeSpaceAllowed Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

mergeSpaceAllowed constraints have a `Value` that represents the maximum distance, in user units, between two geometries for which you are allowed to merge the geometries.

Related Topics

Spacing Constraints

check_space

minBoundaryInteriorHalo

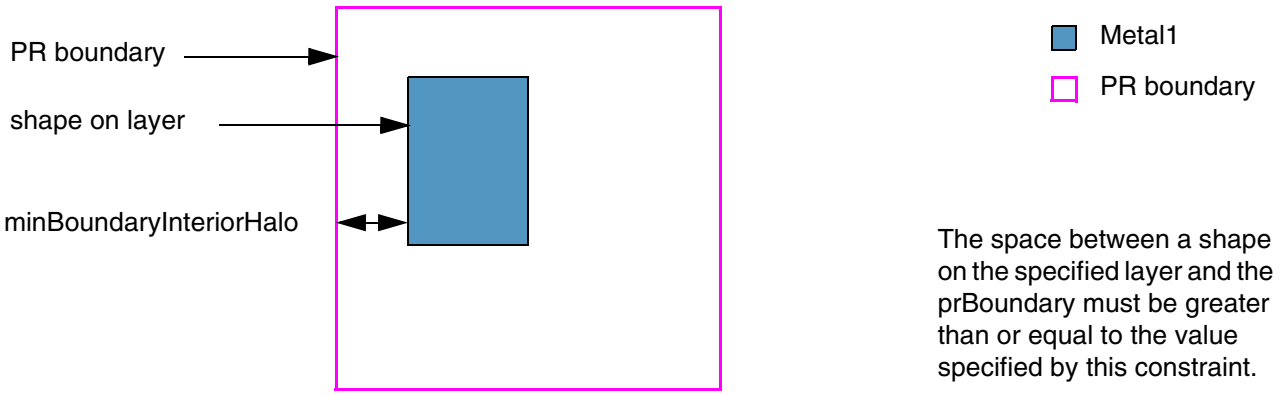
Sets the minimum spacing required between the edges of shapes on the specified layer and the PR boundary. You can optionally limit the constraint to consider only vertical or only horizontal edges and specify an incremental step size for the halo spacing which must be met for the constraint to be satisfied.

minBoundaryInteriorHalo Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

Value	Specifies in user units the minimum distance required between the edge of the shape and the PR boundary.



Optional Parameters

`coincidentAllowed`

Specifies whether shapes must meet the minimum distance (`false`, default) or their edges can be coincident with the boundary (`true`). *This parameter is not currently supported by Space-based Router and Chip Optimizer*.

Type: `BoolValue`

`interSpace`

Restricts the size of the halo to integer multiples of this value.

Type: `IntValue`

`oaSpacingDirection`

Specifies the extension direction to which the constraint applies.

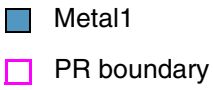
Type: `IntValue`
0 any
1 horizontal
2 vertical

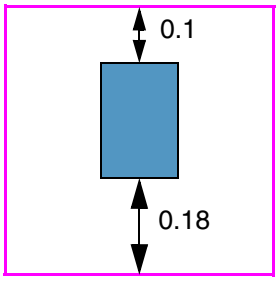
Examples

```
set_constraint_parameter -name oaSpacingDirection -IntValue 2
set_constraint_parameter -name interSpace -IntValue 0.04
set_layer_constraint -constraint minBoundaryInteriorHalo \
  -layer Metal1 -Value 0.1
```

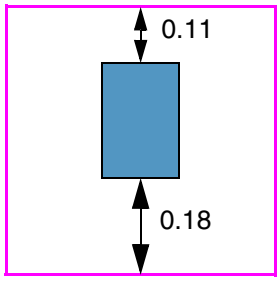
Sets the minimum spacing between the vertical extensions of `Metal1` shapes and the PR boundary to 0.1+(0.04*x) user units (where x is a positive integer).

```
minBoundaryInteriorHalo  0.1
  oaSpacingDirection     2 (vertical)
  interSpace             0.04
```

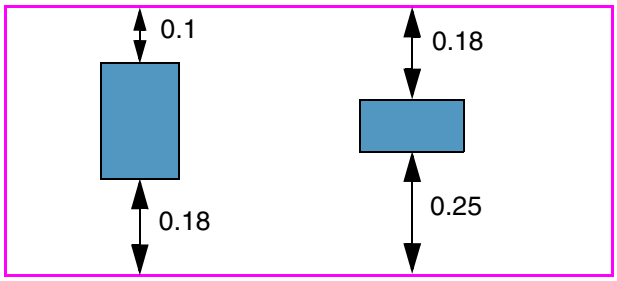




a) PASS. Both vertical extensions are>= to 0.1 and are equal to 0.1 plus an exact multiple of 0.04.



b) FAIL. The 0.11 extension value is not equal to the constraint value of 0.1 plus an exact multiple of 0.04.



c) FAIL. The 0.25 extension value is not equal to the constraint value of 0.1 plus an exact multiple of 0.04.

Related Topics

Spacing Constraints

check_space

minCenterToCenterSpacing

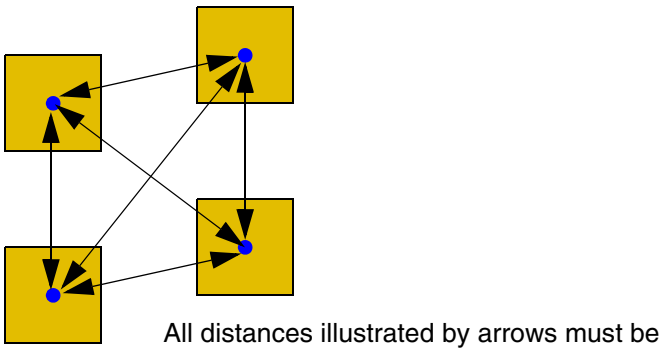
Sets the minimum spacing, center-to-center, between shapes on a single cut layer. This constraint is used to specify the spacing between adjacent vias. An alternate way to specify spacing constraints for adjacent vias is using minAdjacentViaSpacing.

minCenterToCenterSpacing Quick Reference

Constraint Type	Layer (cut layers only)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`Value`	Specifies the minimum distance required between the vias, in user units. The distance is measured from the center of the first cut shape to the center of the second cut shape.



The spacing between the centers of two cut shapes on the specified layer must be greater than or equal to the value specified by this constraint.

Examples

Sets the minimum center-to-center spacing on `VIA2` to `0.3`.

```
set_layer_constraint -constraint minCenterToCenterSpacing -layer VIA2 -Value 0.3
```

Related Topics

Spacing Constraints

minClusterSpacing

Specifies the minimum spacing between two groups of aligned shapes or a group of aligned shapes and another shape on the same layer. The constraint specifies several conditions for the neighboring shapes to be considered as a valid group of shapes.

minClusterSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`Value`	Specifies the minimum spacing between two clusters or a cluster and a neighboring shape on the same layer. If the constraint value is set to zero, the existence of the cluster itself is a violation. As a result, spacing between the cluster and neighboring shapes is not checked.

Required Parameters

The conditions specified by the following parameters must be satisfied for a cluster to be valid:

`shapeWidthRange`	Specifies that a valid cluster is formed only if all its member shapes have widths in this range. Type: `RangeValue`, in user units
`numShapesRange`	Specifies that a valid cluster is formed only if the number of shapes in the cluster is in this range. Type: `RangeValue`

`spacingRange`	Specifies that a valid cluster is formed only if the neighboring shapes in a cluster are within this centerline spacing range. The spacing between the centerline of two shapes must be less than or equal to this value. Type: `RangeValue`, in user units

Optional Parameters

`groupDirection`	Specifies that the groups can only be formed in the given direction. Type: `IntValue` 0 anyDirection (horizontally and vertically, default) 1 horizontalDirection (horizontally only) 2 verticalDirection (vertically only)
`centerLineDistance`	Specifies that the spacing measurement is done from the centerline of the cluster to the centerline of another shape. If this parameter is false, the spacing measurement is done from the edge of the cluster to the edge of another shape. **Note:** The intra-cluster spacing (specified by the parameter `spacingRange)` is always measured centerline. Type: `BoolValue`
`minSpaceOverride`	Specifies that the constraint checks for minimum spacing between the shapes which are not part of any cluster. Type: `BoolValue`
`oaSpacingDirection`	Specifies the measurement direction. Type: `StringAsIntValue` anyDirection 0 Horizontally and vertically (default) horizontalDirection 1 Horizontally only verticalDirection 2 Vertically only
`otherWidthRange`	Specifies that the constraint applies if the width of the other shape is in this range. Type: `RangeArrayValue`, in user units

`parallelRunLength`	Specifies that the constraint applies only if the parallel run length between the cluster and the other shape is equal to or greater than this value. Type: `Value`, in user units
`clusterToCluster`	The constraint applies only between neighboring clusters. Otherwise, the constraint also applies between a cluster and neighboring non-cluster shapes. Type: `BoolValue`
`edgeToEdge`	The spacing between neighboring shapes in a cluster is measured edge-to-edge. Otherwise, the spacing is measured between centerlines. **Note:** This parameter applies only when measuring spacing between neighboring shapes to identify clusters. The `'centerLineDistance` parameter determines how spacing is measured between a cluster and neighboring non-cluster shapes. Type: `BoolValue`
`sameMask`	Clusters are formed between shapes of the same color. Additionally, the constraint applies to neighboring non-cluster shapes only is they are of the same color as the shapes that form the cluster. Type: `BoolValue`

Examples

- Example 1: minClusterSpacing
- Example 2: minClusterSpacing with centerLineDistance
- Example 3: minClusterSpacing with groupDirection
- Example 4: minClusterSpacing with minSpaceOverride

Example 1: minClusterSpacing

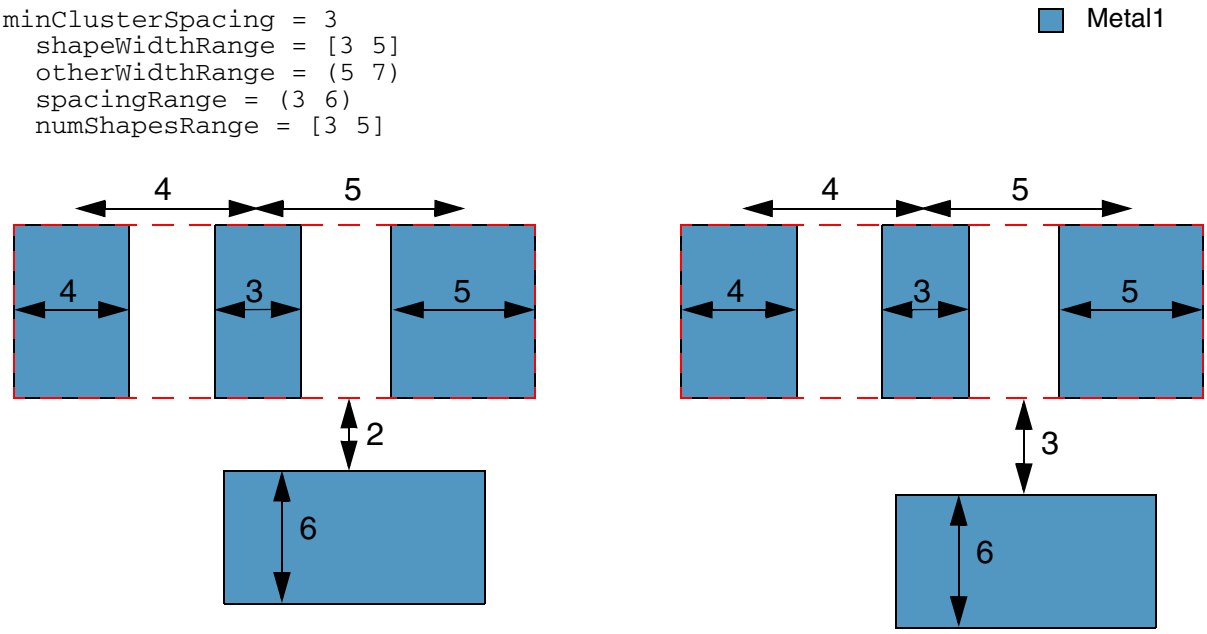
```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[3 5]"}
set_layer_constraint -constraint minClusterSpacing -Value 3 -layer Metal1
```

Specifies that the minimum spacing between the cluster and the other shape must be greater than or equal to 3 user units under the following conditions:

- Widths of shapes in the cluster are greater than or equal to 3 and less than or equal to 5.
- Width of the other shape is greater than 5 and less than 7.
- Centerline spacing between the shapes in the cluster is greater than 3 and less than 6.
- Number of shapes in the cluster is greater than or equal to 3 and less than or equal to 5.

Illustration for minClusterSpacing

```
minClusterSpacing = 3
  shapeWidthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [3 5]
```



a) FAIL. The spacing between the cluster and the other shape is less than 3.

b) PASS. The spacing between the cluster and the other shape is greater than or equal to 3.

Example 2: minClusterSpacing with centerLineDistance

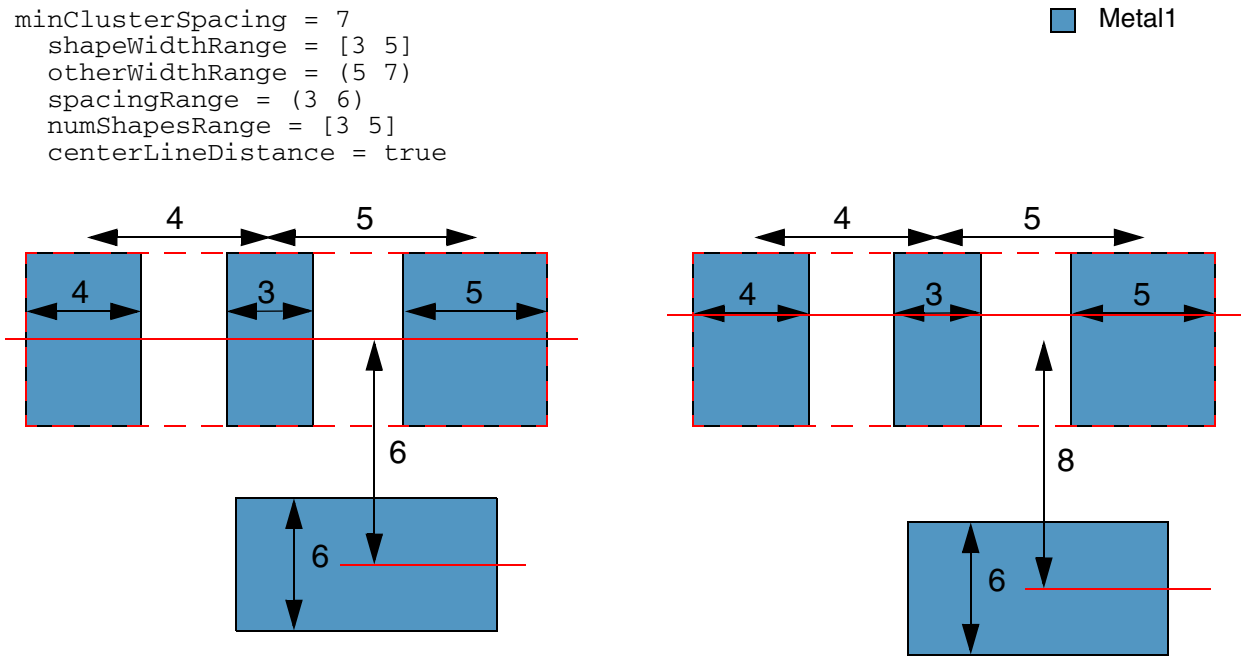
```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[3 5]"}
set_constraint_parameter -name centerLineDistance -BoolValue true
set_layer_constraint -constraint minClusterSpacing -Value 7 -layer Metal1
```

Specifies that the minimum spacing between the cluster and the other shape must be greater than or equal to 7 user units under the following conditions:

- Widths of the shapes in the cluster are greater than or equal to 3 and less than or equal to 5.
- Width of the other shape is greater than 5 and less than 7.
- Centerline spacing between the shapes in the cluster is greater than 3 and less than 6.
- Number of shapes in the cluster is greater than or equal to 3 and less than or equal to 5.
- Spacing measured from the centerline of the cluster to the centerline of the other shape.

Illustration for minClusterSpacing with centerLineDistance

```
minClusterSpacing = 7
  shapeWidthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [3 5]
  centerLineDistance = true
```



a) FAIL. The spacing between the cluster and the other shape is less than 7.

b) PASS. The spacing between the cluster and the other shape is greater than or equal to 7.

Example 3: minClusterSpacing with groupDirection

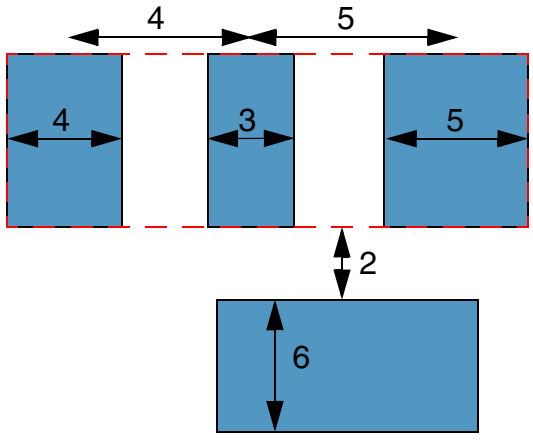
```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[3 5]"}
set_constraint_parameter -name groupDirection -IntValue 2
set_layer_constraint -constraint minClusterSpacing -Value 5 -layer Metal1
```

Specifies that the minimum spacing between the cluster and the other shape must be greater than or equal to 5user units under the following conditions:

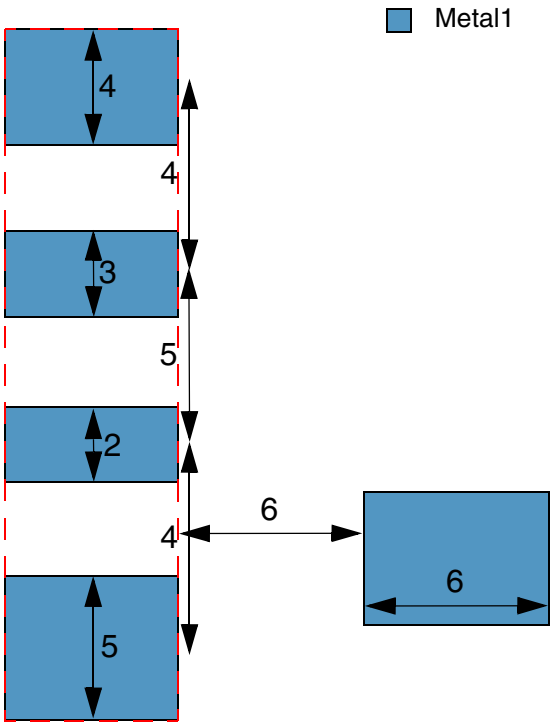
- Widths of the shapes in the cluster are greater than or equal to 3 and less than or equal to 5.
- Width of the other shape is greater than 5 and less than 7.
- Centerline spacing between the shapes in the cluster is greater than 3 and less than 6.
- Number of shapes in the cluster is greater than or equal to 3 and less than or equal to 5.
- Groups are formed in the vertical direction.

Illustration for minClusterSpacing with groupDirection

```
minClusterSpacing = 5
  widthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [3 5]
  groupDirection = vertical
```



a) Constraint does not apply. The group direction is not vertical.



b) PASS. The spacing between the vertical cluster and the other shape is greater than 5.

Example 4: minClusterSpacing with minSpaceOverride

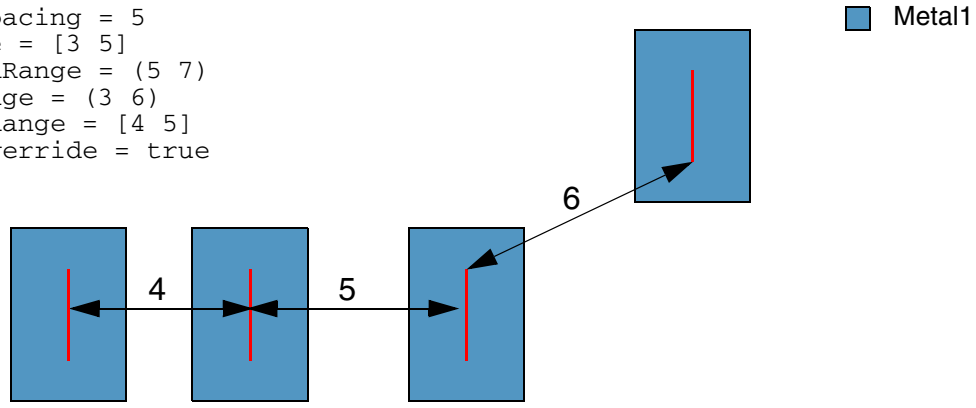
```
set_constraint_parameter -name shapeWidthRange -RangeValue {"[3 5]"}
set_constraint_parameter -name otherWidthRange -RangeValue {"(5 7)"}
set_constraint_parameter -name spacingRange -RangeValue {"(3 6)"}
set_constraint_parameter -name numShapesRange -RangeValue {"[4 5]"}
set_constraint_parameter -name minSpaceOverride -BoolValue true
set_layer_constraint -constraint minClusterSpacing -Value 5 -layer Metal1
```

Specifies that the minimum spacing between the cluster and the other shape must be greater than or equal to 5 user units under the following conditions:

- Widths of shapes in the cluster are greater than or equal to 3 and less than or equal to 5.
- Width of the other shape is greater than 5µm and less than 7µm.
- Centerline spacing between the shapes in the cluster is greater than 3 and less than 6.
- Number of shapes in the cluster is greater than or equal to 4 and less than or equal to 5.
- Constraint applies as minSpacing to the shapes which are not part of any cluster.

Illustration for minClusterSpacing with minSpaceOverride

```
minClusterSpacing = 5
  widthRange = [3 5]
  otherWidthRange = (5 7)
  spacingRange = (3 6)
  numShapesRange = [4 5]
  minSpaceOverride = true
```



a) FAIL. The constraint applies as minimum spacing for all the non-cluster shapes but is violated because the spacing between the leftmost shapes is only 4, less than the minimum required spacing of 5.

Related Topics

Spacing Constraints

check_space

minClusterSpacing

minDiagonalSpacing

Sets the minimum spacing, edge-to-edge, between diagonal shapes on a layer.

minDiagonalSpacing Quick Reference

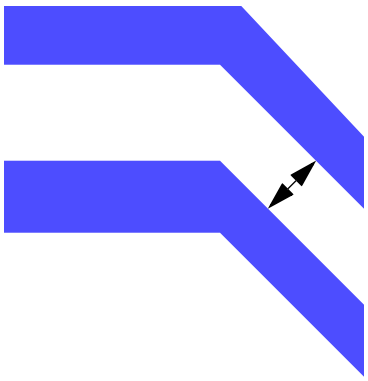
Constraint Type	Layer
Value Types	`Value`, `OneDTblValue`, `TwoDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Types

`minDiagonalSpacing` constraints have the following value types:

- `Value`

  Specifies the minimum spacing, edge-to-edge, between diagonal shapes on the specified layer.

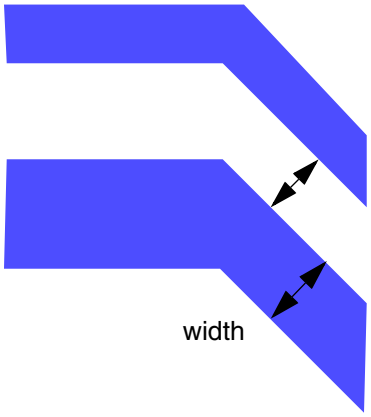


Fixed Value
The spacing between two diagonal shapes on the specified layer must be greater than or equal to the value specified by this constraint.

- `OneDTblValue`

  Specifies the minimum spacing, edge-to-edge, between diagonal shapes on the specified layer as a 1D table. The table value is the minimum spacing, based on the

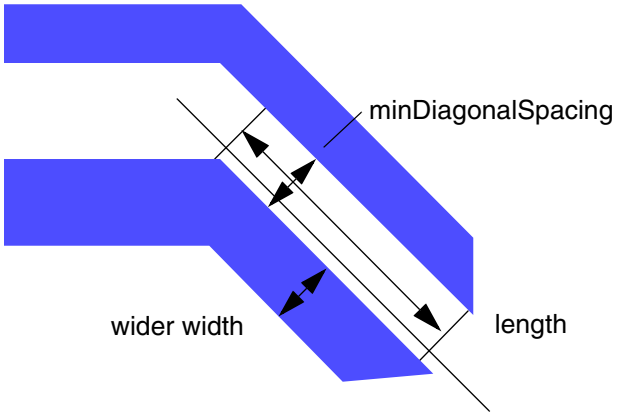
lookup key (`width`) which is the width of the wider of the two shapes. The width of a shape is defined as the smaller of the shape's two dimensions.



1D Table
The minimum required spacing between two diagonal shapes on the specified layer varies, based on the width of the wider of the two shapes.

- `TwoDTblValue`

  Specifies the minimum spacing, edge-to-edge, between diagonal shapes on the specified layer as a 2D table. The table value is the minimum spacing, based on the row lookup key (`width`) as the width of the wider of the two shapes, and the column lookup key (`length`) as the parallel run length between the two shapes. The parallel run length is calculated as the distance in the lengthwise direction for which the two shapes are directly orthogonal such that a perpendicular line to the length would intersect both shapes.



2D Table
The minimum required spacing between two diagonal shapes on the specified layer varies, based on the width of the wider of the two shapes and the parallel run length between the shapes.

Parameters

- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1). For information on this parameter, see Euclidean and Manhattan Spacing Constraints.

- `widthLengthTableType` (`IntValue`) specifies the table type when `minSameNetSpacing` is a `TwoDTblValue` table. The values are given in Table 16-1 on page 811.
- `eolAddSpace` (`OneDTblValue`, optional) specifies additional spacing required based on the wire width.
- `eolRadialCheck` (`BoolValue`, optional) enables radial checking from the wire corners.

Related Topics

Spacing Constraints

check_space

minDiffIslandParallelViaSpacing

This constraint is obsolete. Use minParallelViaSpacing instead.

minDiffPotentialSpacing

Sets the minimum required distance between two objects at different voltage levels on the same layer.

minDiffPotentialSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minDiffPotentialSpacing` constraints have a `Value` that represents the minimum required distance between two objects at different voltage levels on the same layer.

Related Topics

Spacing Constraints

minEdgeLengthSpacing

Specifies the minimum spacing between a shape on the given layer and the edges of a second shape on the same layer, which are less than a given length. Optionally, this constraint applies if the shapes share a common run length that is greater than or equal to a specified length. Also, the minimum edge length spacing can be optionally restricted to edges that are in a particular direction.

minEdgeLengthSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

Value	Specifies the minimum spacing to short edges.

Required Parameter

length	The constraint applies only to edges shorter than this length. Type: Value

Optional Parameters

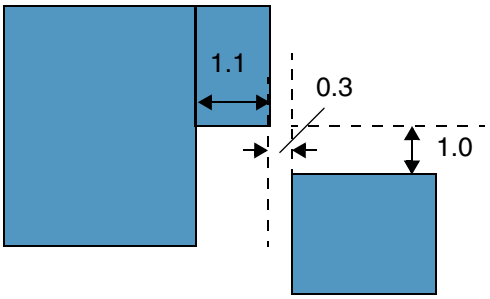
parallelRunLength	Specifies that the constraint applies only edges where the parallel run is at least this length in user units. Type: Value
edgeDirection	Specifies the edge length measurement direction. Type: IntValue 0 any 1 horizontal 2 vertical

Examples

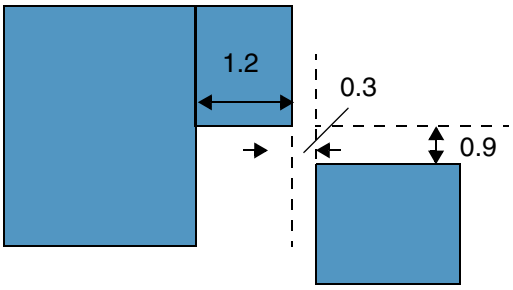
```
set_constraint_parameter -name length -Value 1.2
set_constraint_parameter -name parallelRunLength -Value -0.3
set_constraint_parameter -name edgeDirection -IntValue 1
set_layerpair_constraint -constraint minEdgeLengthSpacing \
  -layer Metal1 -Value 1.0
```

Specifies that horizontal edges on `Metal1` shorter than 1.2 user units must be spaced at least 1.0 user units from other `Metal1` edges with a parallel run length greater than or equal to -0.3.

Illustration for minEdgeLengthSpacing



a) PASS. Constraint applies and is met.



b) PASS. Constraint does not apply because there is no horizontal edge that is less than 1.2 user units long.

Related Topics

Spacing Constraints

minEnclosedSpacing

Defines the minimum distance, parallel edge-to-edge, across a hole in the specified layer.

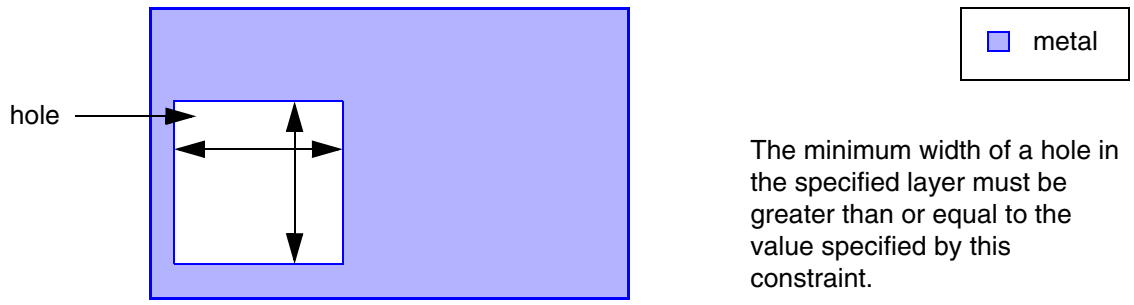
minEnclosedSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`, `OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Types

- `Value`

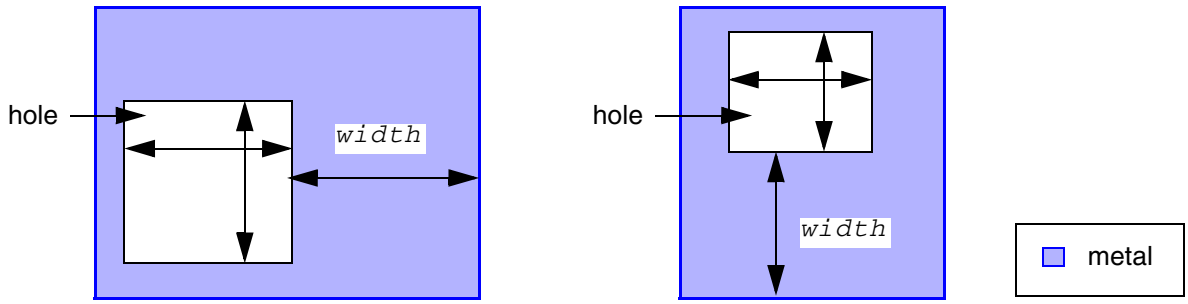
  Specifies the minimum width, parallel edge-to-edge, in the X and Y directions for a hole on the layer.



- `OneDTblValue`

  Specifies the minimum width, parallel edge-to-edge, in the X and Y directions for a hole on the layer, based on the width of the enclosing metal. The lookup key (`width`) for the

table is the largest effective width of the enclosing metal, measured from the edge of the hole. The table value is the minimum width for the hole.



Related Topics

Spacing Constraints

minEndOfLineCutSpacing

Specifies the spacing rules between cuts near the ends of lines. This increased spacing is required when neighbor metal lines intersect with the regions defined on either side of the line. The cut must have specific metal enclosures. The required spacing is dependent on the parallel run length (prl) with neighbor cuts.

It is a layer-pair constraint, where the second layer should be the metal layer above the cut layer. In addition, the placement of the cut within that layer governs the spacing of the cut.

In some processes, the spacing can also be specific to the cut class of each cut. Therefore, it is necessary to specify a list of cut classes and the required spacing from the named cut class to each of the cut classes in the list.

minEndOfLineCutSpacing Quick Reference

Constraint Type	Layer-pair (non-symmetric)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Types

`minEndOfLineCutSpacing` constraint has the following value type:

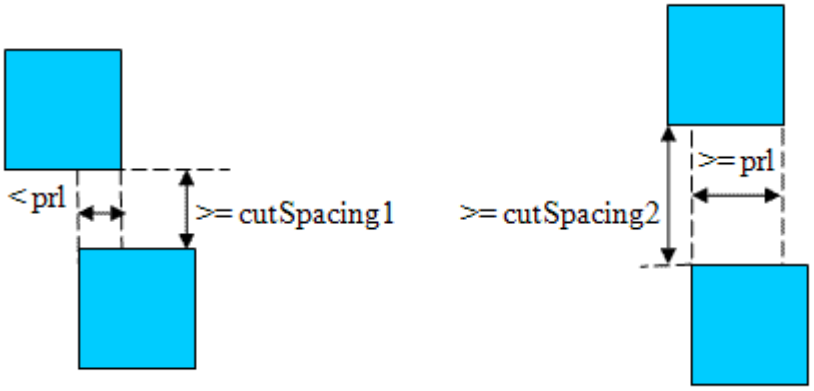
- `Value`

Parameters

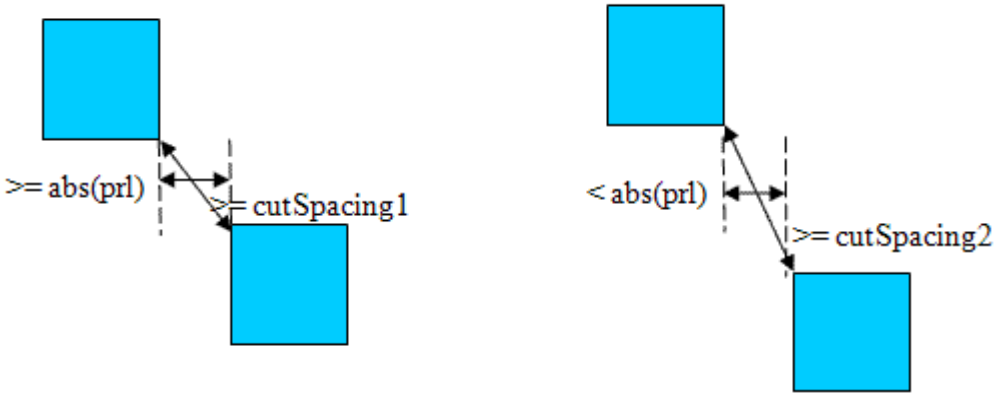
- `cutClass` (optional) specifies the dimensions of the cut layer's cut class if this constraint is specific to a particular cut class.
- `width` (required) specifies that the rule only applies if the width of the end-of-line is less than the given width.
- `parallelRunLength` (required) If the parallel run length between the two cuts is `< prl`, `cutSpacing1` is required. However, if the parallel run length is `>= prl`, `cutSpacing2` is required.

  Note: `cutSpacing1` is typically smaller than `cutSpacing2`. Positive and negative

values are allowed.



Example of positive parallel run length (prl)



Example of negative parallel run length (prl)

- `enclosure` (required) The two values of `enclosure` are `eolEnclosure` and `otherEnclosure`. The rule only applies if the enclosure on the end-of-line edge is < `eolEnclosure` and the enclosure on one of the other cut edges orthogonal to the end-of-line edge is `otherEnclosure`.
- `extension` (required) specifies a search region along the wire, defined by extending `sideExt` along the end-of-line edge, away from the wire, and `backwardExt` along the orthogonal edge, where the cut has an enclosure equal to `otherEnclosure`.
- `spanLength` (required) The rule applies if the wire containing the cut has a span length `>= spanLength`, then a neighbor wire must overlap with the search region along the

orthogonal edge where the cut has enclosure = `otherEnclosure`. However, if the wire containing the code has a span length < `spanLength`, then the rule only applies if the following two conditions apply:

- The neighbor wire overlaps the search region along the exact enclosure edge.
- No neighbor wire overlaps the search region along the edge opposite the exact enclosure edge.

Examples

- **Example 1—minEndOfLineCutSpacing example**
- Example 2—minEndOfLineCutSpacing example

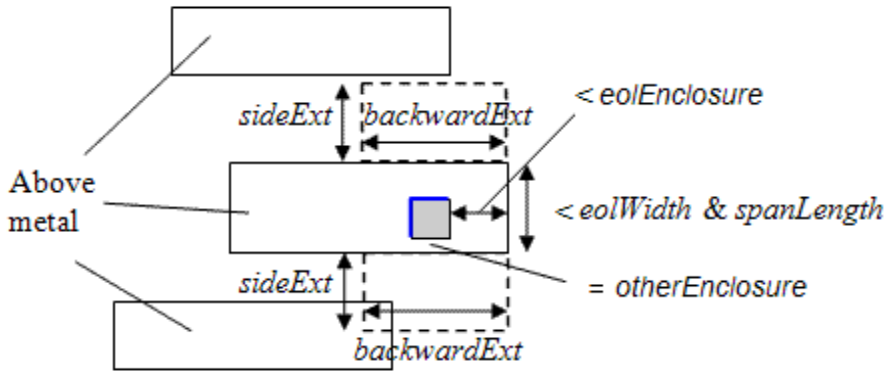
Example 1—minEndOfLineCutSpacing example

In this example, the `minEndOfLineCutSpacing` rule would apply under the following conditions:

- The line width is < `eolWidth`.
- The end-of-line enclosure is < `eolOverhang` and the enclosure of the bottom cut edge is `otherOverhang`.
- The span length is < `spanLength` and there is a neighbor wire in the search region along the exact `otherOverhang` enclosure edge, and there is no neighbor wire in the search region on the opposite side of the wire from the edge with the exact `otherOverhang` enclosure.

The cut spacing will apply to the vertical blue edge because it is opposite to the end of the line. It will apply to the horizontal blue edge because it is opposite to the edge with the exact `otherOverhang` enclosure.

Illustration for minEndOfLineCutSpacing



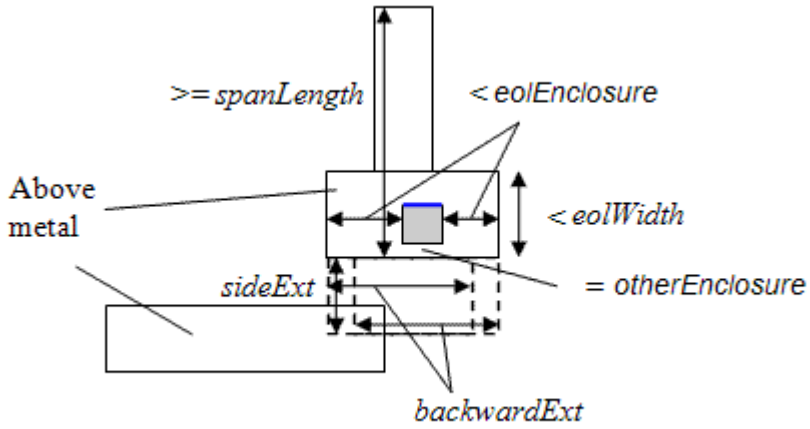
Example 2—minEndOfLineCutSpacing example

In this example, the `minEndOfLineCutSpacing` rule would apply under the following conditions:

- The line width is `< eolWidth`.
- The end-of-line enclosure is `< eolOverhang` and the enclosure of the bottom cut edge is `otherOverhang`.
- The span length is `>= spanLength` and there is a neighbor wire in the search region along the exact `otherOverhang` enclosure edge.

Since the cut is within `eolOverhang` of both end-of-line edges of the horizontal wire, there are two ends-of-line. This implies that there are two search regions, one extending `backwardExt` to the left of the rightmost edge, and the other one extending `backwardExt` to the right of the leftmost edge. It also implies that there is no cut edge opposite the end-of-line edge to which the cut spacing applies (both cut edge are end-of-line edges). Therefore, the only edge to which the cut spacing applies is the blue edge, opposite to the edge with the exact `otherOverhang` enclosure.

Illustration for minEndOfLineCutSpacing



Related Topics

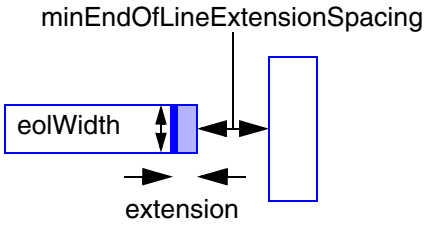
check_layerpair_space

minEndOfLineCutSpacing

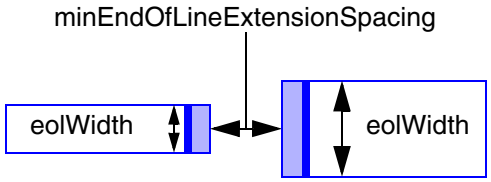
minEndOfLineExtensionSpacing

Specifies the minimum spacing between an end-of-line edge of a given width and neighboring wire edges when a specified extension must be applied to the end-of-line before the spacing is checked.The constraint can also specify an extension that must be applied to the neighbor wire, when the neighbor wire is an end-of-line of a given width, before spacing is checked. You can use optional parameters to specify when the constraint does not apply, based on the length of the sides of the wire adjacent to the end-of-line edge.

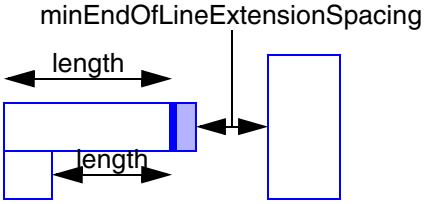
minEndOfLineExtensionSpacing Descriptions



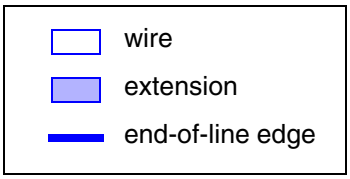
end-to-side
Example of minEndOfLineExtensionSpacing where the neighboring edge is the side of edge of a wire. The rule specifies an extension that must be applied to the end-of-line before the spacing is checked.



end-to-end
Example of minEndOfLineExtensionSpacing where the neighboring edge is also an end-of-line. In these cases, the rule can specify extensions that must be applied to both end-of-lines before the spacing is checked.



Optional `length` and `twoSides` parameters specify when the constraint applies based on the length of the sides that are adjacent to the end-of-line edge.



minEndOfLineExtensionSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

The `minEndOfLineExtensionSpacing` constraint has a `Value` that represents the spacing in user units.

Parameters

- `extensionSpacing` (`Dual1DTblValue`, required) is two 1-D tables of values. The first table has end-of-line width (*eolWidth*)/extension pairs for end-to-side cases and the second table has *eolWidth*/extension pairs for end-to-end cases.

  For each end-of-line, find the last pair for which the end-of-line width is less than (<) *eolWidth*, in user units, to determine the extension that must be applied to the end-of-line edge before spacing is checked.

- `length` (`Value`, optional) with the `twoSides` parameter, specifies when the constraint applies based on the side lengths of the end-of-line.

- `twoSides` (`BoolValue`, optional) applies only when the `length` parameter is given and determines when the constraint applies as follows:

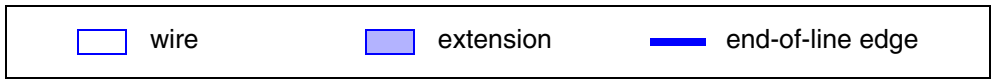
`length` with `twoSides false` or not given	The constraint applies when at least one side of the end-of-line is greater than or equal to `length`. (The constraint does not apply if both sides of the end-of-line are less than `length`.)
`length` with `twoSides true`	The constraint applies when both sides of the end-of line are greater than or equal to `length`. (The constraint does not apply if either side of the end-of-line is less than `length`).

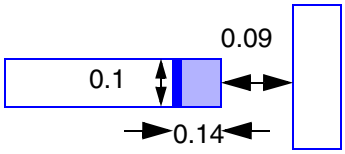
Examples

In this example, the minimum spacing between an end-of-line and a neighboring wire is 0.1 when extensions are applied under the following conditions:

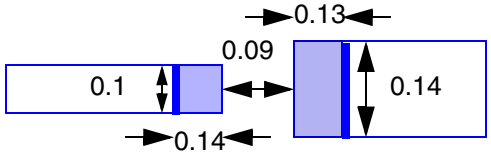
Case type	For end-of-line width:	Apply this extension to the end-of-line before checking spacing:
end-to-side	< 0.11	0.14
end-to-side	>= 0.11 and < 0.15	0.12
end-to-end	< 0.11	0.00
end-to-end	>= 0.11 and < 0.15	0.13

minEndOfLineExtensionSpacing with end-to-end extensions

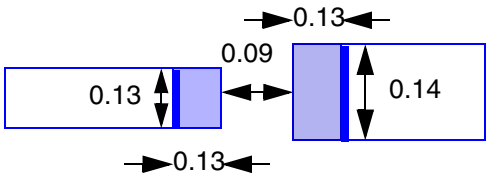




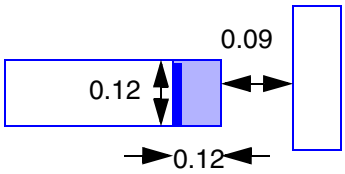
a) Violation. For end-to-side case, 0.14 extension is applied for left end-of-line with width < 0.11, and 0.09 spacing does not meet the required minimum 0.1 spacing after the extension is applied.



b) Violation. For end-to-end case, 0.14 extension is applied for left end-of-line with width < 0.11, and 0.13 extension is applied for right end-of-line with width > 0.11 and < 0.15, and the 0.09 spacing does not meet the required minimum of 0.1.



c) Violation. For end-to-end case, 0.13 extension is applied to both wires with width >= 0.11 and < 0.15, and a minimum 0.1 spacing is required but not met.



d) Violation. For end-to-side case, 0.12 extension is applied for end-of-line with width >= 0.11 and < 0.15, but the 0.09 spacing does not meet the required minimum of 0.1.

Format	Example

Tcl	`set_constraint_parameter -name extensionSpacing \`
	`  -OneDTblValue  {0.11 0.14 0.15 0.12} \`
	`  -OneDTblValueB {0.11 0.00 0.15 0.13}`
	`set_layer_constraint -layer Metal2 \`
	`  -constraint minEndOfLineExtensionSpacing -Value 0.1`

LEF	`EOLEXTENSIONSPACING 0.1`
	`  ENDOFLINE 0.11 EXTENSION 0.14`
	`  ENDOFLINE 0.15 EXTENSION 0.12 ENDTOEND 0.13 ;`

Related Topics

Spacing Constraints

minEndOfLinePerpSpacing

minEndOfLineSpacing

minEndOfLinePerpSpacing

Specifies the minimum distance required between an end-of-line and a perpendicular edge. Typically, `minEndOfLinePerpSpacing` is slightly larger than minSpacing on the layer.

minEndOfLinePerpSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

The `minEndOfLinePerpSpacing` constraint has a `Value` that represents the required minimum distance, in user units, between an end-of-line wire and unobstructed edges that are perpendicular to it.

Format	Example

Tcl	`set_constraint_parameter -name width -Value eolwidth`
	`set_constraint_parameter -name endOfLinePerpWidth -Value perWidth`
	`set_layer_constraint -constraint minEndOfLinePerpSpacing \`
	`  -layer Metal1 -Value eolSpace`

LEF	`PROPERTY LEF58_SPACING` `"SPACING eolSpace EOLPERPENDICULAR eolwidth perWidth ;"`

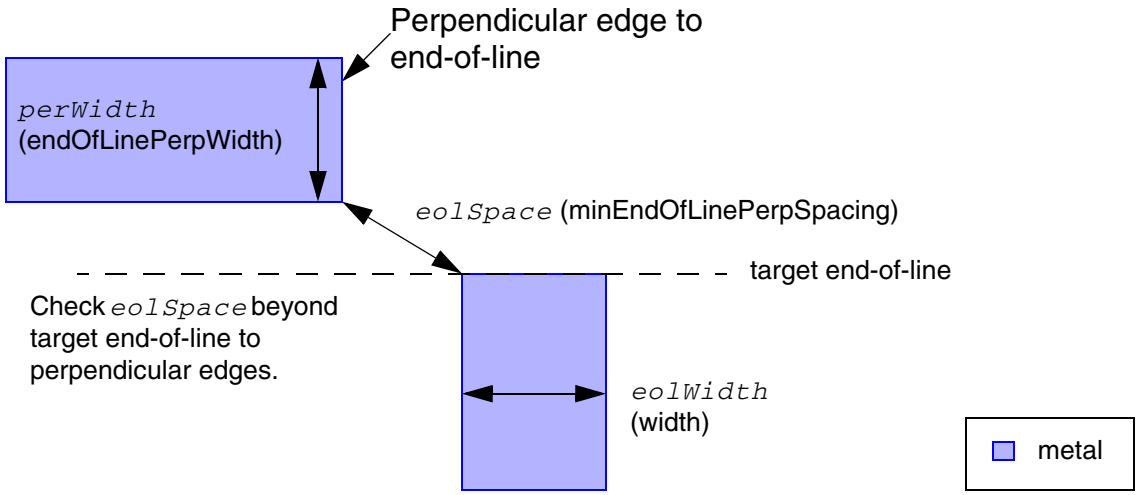
Parameters

The following parameters are required:

- `width` (`Value`) is the width, in user units, of the end-of-line (*`eolWidth`*).
- `endOfLinePerpWidth` (`Value`) is the length, in user units, of the perpendicular edge (*`perWidth`*).

The constraint is triggered when the width of an end-of-line is less than `width`, and an edge that is perpendicular to the end-of-line is less than or equal to `endOfLinePerpWidth` in length. The perpendicular edge must also be *within sight* of the end-of-line, such that an

unobstructed line can be drawn from the end-of-line to the perpendicular edge, not including the endpoints of either edge.



Examples

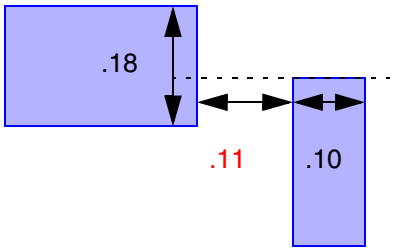
The following example requires a minimum spacing of 0.12 user units between an end-of-line of width less than 0.11 user units and a perpendicular edge that is less than or equal to 0.18 user units in length. In addition, a minimum spacing of 0.11 is set for shapes on the layer.

Format	Example

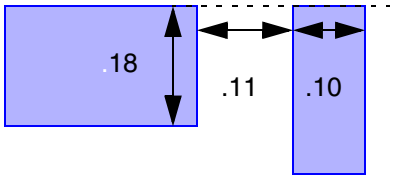
Tcl	`set_constraint_parameter -name width -Value 0.11`
	`set_constraint_parameter -name endOfLinePerpWidth -Value 0.18`
	`set_layer_constraint -constraint minEndOfLinePerpSpacing \`
	`  -layer Metal1 -Value 0.12`

LEF	`PROPERTY LEF58_SPACING` `"SPACING 0.12 EOLPERPENDICULAR 0.11 0.18 ;"`

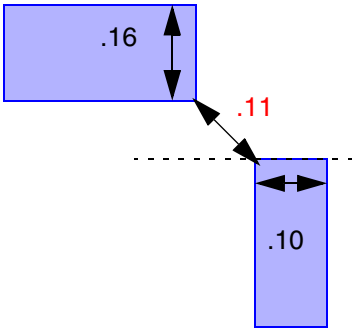
The following figures show how the constraint is interpreted in this example.



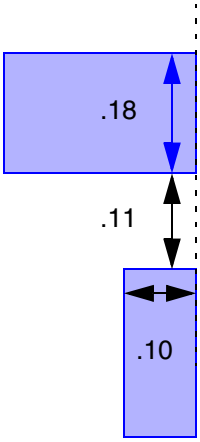
a) Violation,
end-of-line is < 0.11 wide,
perpendicular edge = 0.18, and
is < 0.12 from EOL edge.



b) No Violation,
perpendicular edge is not *seen*
by end-of-line, so it is not
checked.



c) Violation,
end-of-line is < 0.11 wide,
perpendicular edge < 0.18, and
is < 0.12 from EOL edge.



d) No Violation,
perpendicular edge is not *seen*
by end-of-line, so it is not
checked.

Related Topics

Spacing Constraints

minEndOfLineExtensionSpacing

minEndOfLineSpacing

minEndOfLineSpacing

Specifies the minimum spacing between the end of a line and its neighboring geometry. The constraint definition supports optional parameters to specify when the constraint applies.

- The constraint applies only when the width of the end-of-line wire is less than a specified value (`width` parameter) or within a specified range (`widthRangeArray` parameter).
- The constraint applies when any geometry occurs within a region defined by the minimum end-of-line spacing (`minEndOfLineSpacing` constraint), the distance (`distance` parameter) from each side of the wire and the width of the wire.
- The constraint applies only if one parallel edge is within a specified rectangular region from the corners of the end-of-line wire, or it applies only if two parallel edges are within a specified rectangular region from the corners of the end-of-line wire. The dimensions of the rectangular region are given by the distance perpendicular to the wire (`parallelEdgeSpace` parameter), the distance from the end-of-line along the wire (`parallelEdgeWithin` parameter), and the lateral verification distance (`distance` parameter).
- The constraint applies when no parallel edges occur within the region defined by the minimum end-of-line spacing, or when one parallel edge occurs within the region defined by the minimum end-of-line spacing, or when two parallel edges occur within the region defined by the minimum end-of-line spacing.

minEndOfLineSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing
Group Operators	AND, OR

Value Type

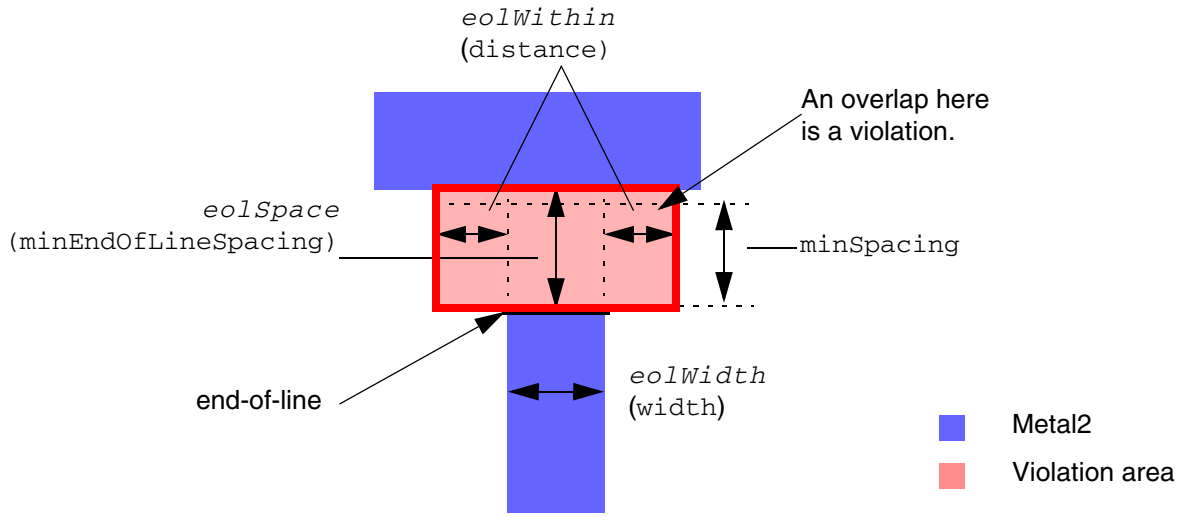
`minEndOfLineSpacing` constraints have a `Value` that represents the minimum spacing between the end of a line and its neighboring geometry. Typically, the `minEndOfLineSpacing` is slightly larger than the minimum required spacing (minSpacing).

The application of the End-of-Line Spacing rule is dependent on the given `count` parameter:

- Case 1—When the `count` parameter is 0

In this case, an end-of-line with a width less than *`eolWidth`* requires spacing greater than or equal to *`eolSpace`* beyond the end-of-line anywhere within (that is, less than) *`eolWithin`* distance.

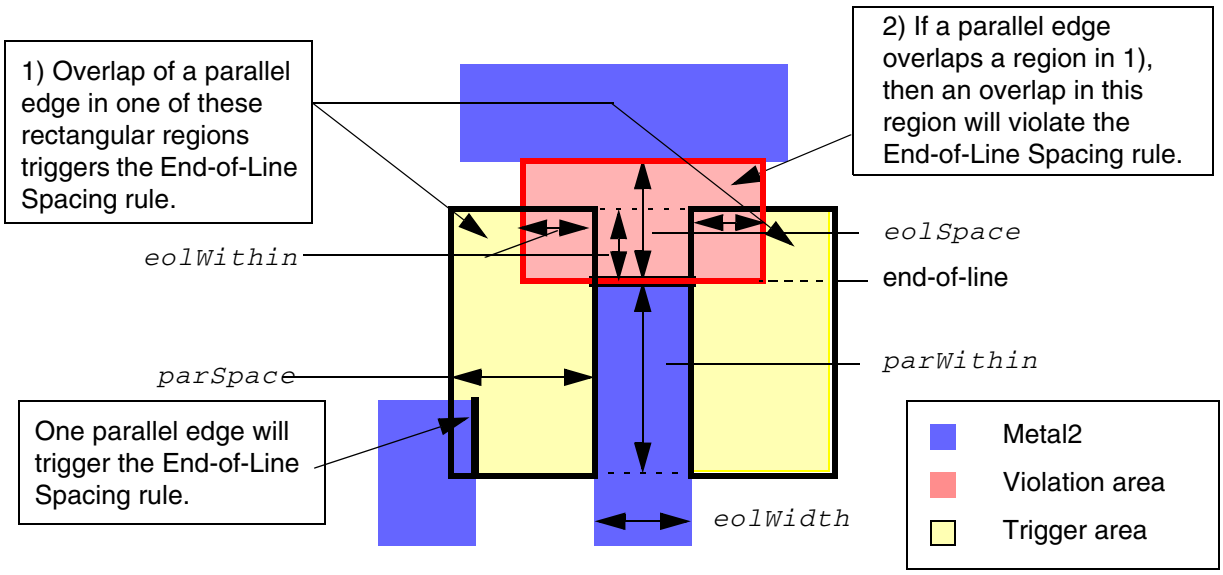
minEndOfLineSpacing with count = 0



- Case 2—When the `count` parameter is 1

In this case, the end-of-line rule applies only if there is one parallel edge that is less than *`parSpace`* away, and is also less than *`parWithin`* from the end-of-line and *`eolWithin`* beyond the end-of-line.

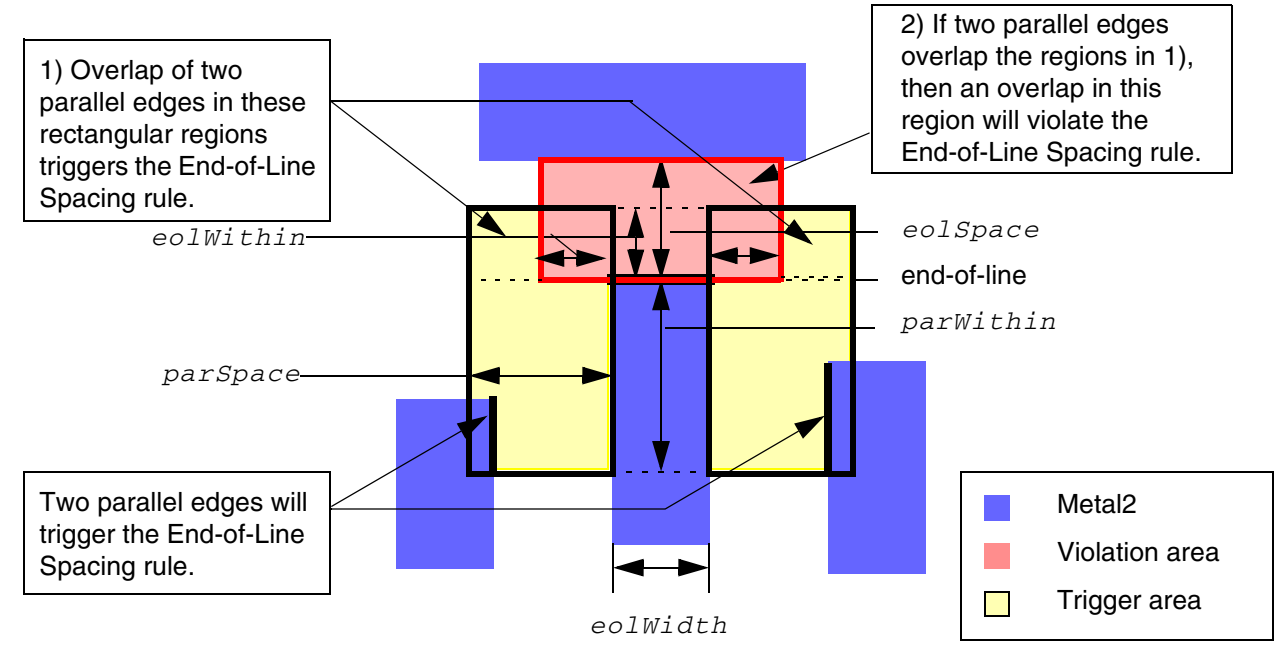
minEndOfLineSpacing with count = 1



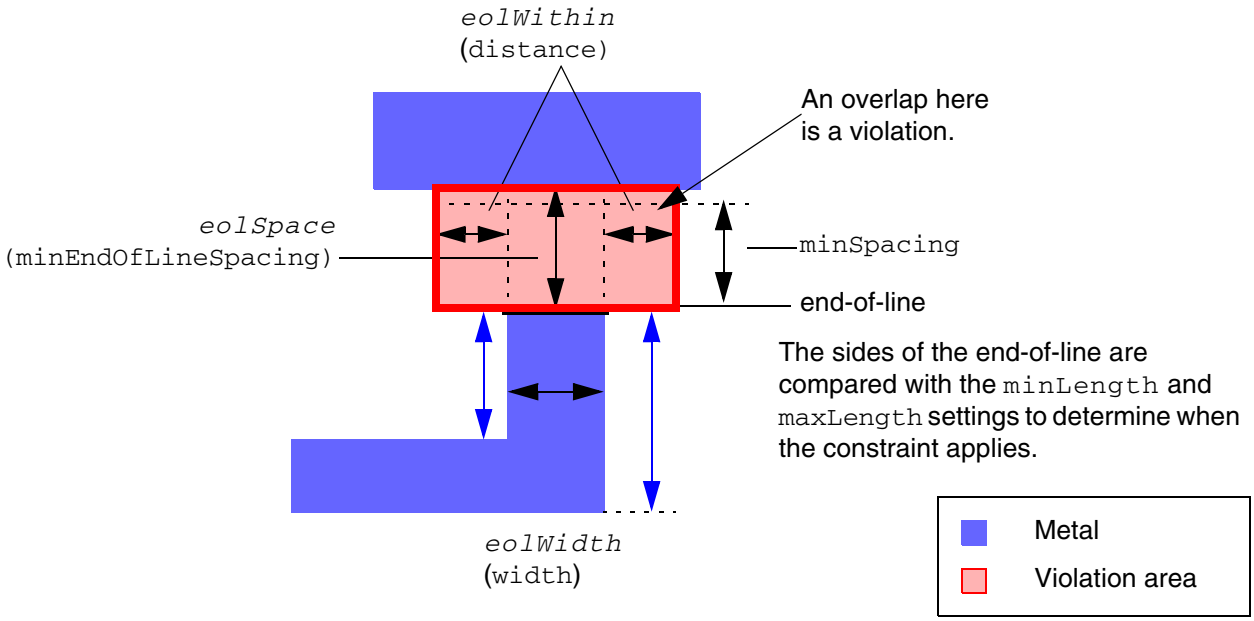
- Case 3—When the `count` parameter is 2

In this case, the End-of-Line Spacing rule is triggered if there are two parallel edges that are less than `parallelEdgeSpace` from the end-of-line wire within the regions given by the `distance` and the `parallelEdgeWithin` parameters as shown in the following figure.

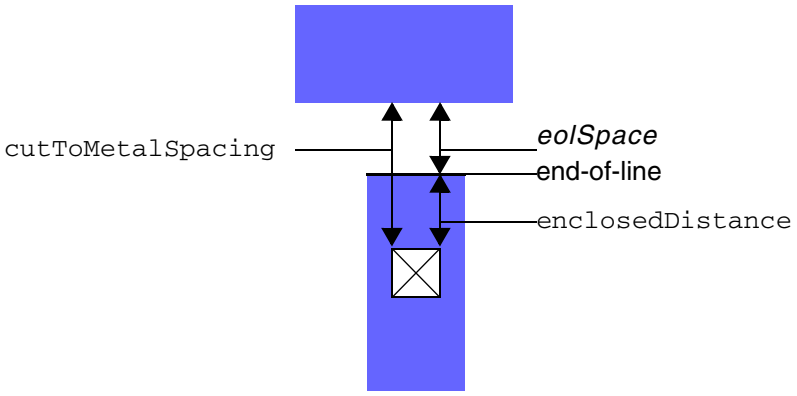
minEndOfLineSpacing with count = 2



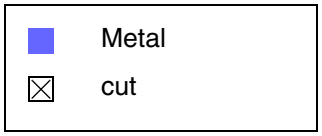
minEndOfLineSpacing with minLength and/or maxLength



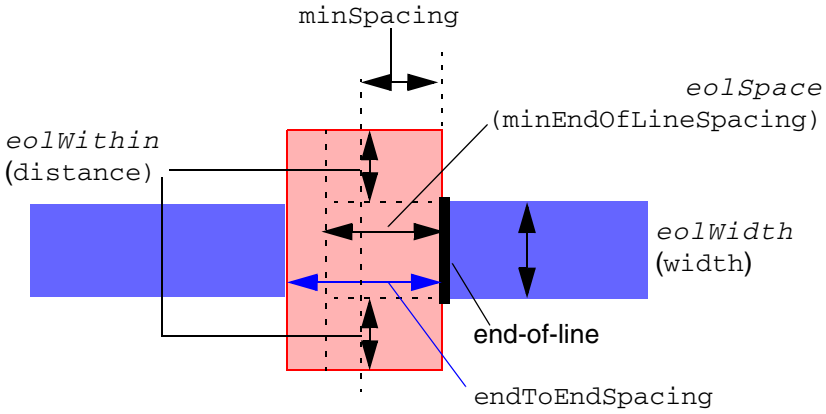
minEndOfLineSpacing with cutToMetalSpacing



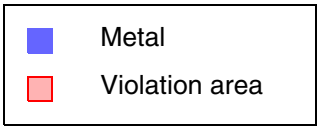
The constraint applies when the distance between the EOL edge and a cut edge <`enclosedDistance` **and** the cut edge is < `cutToMetalSpacing` from the metal beyond the EOL edge.



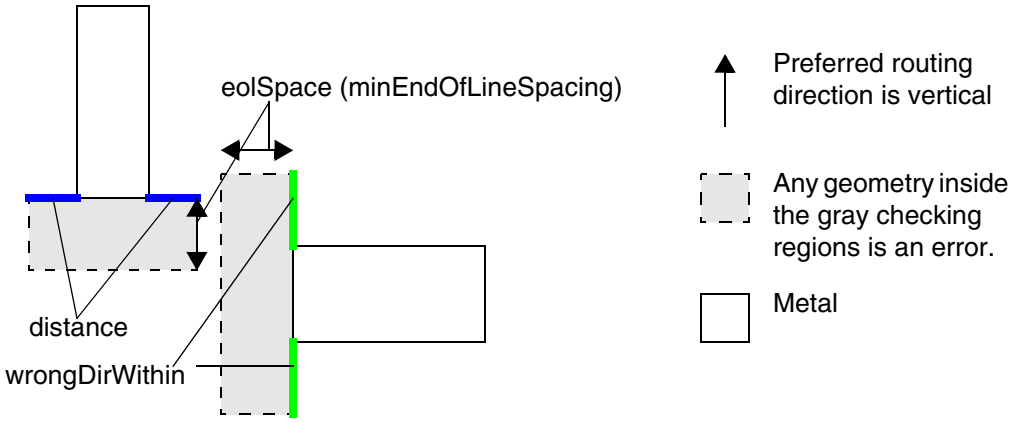
minEndOfLineSpacing with endToEndSpacing



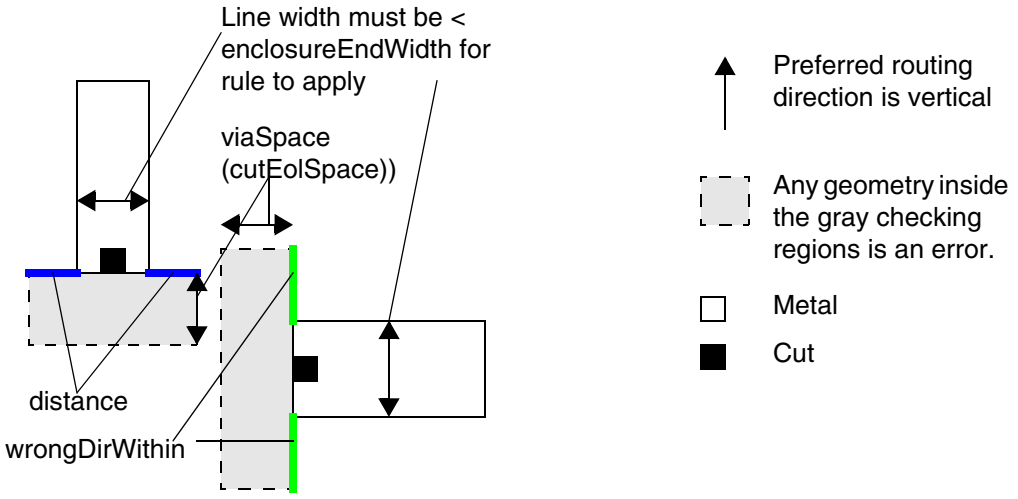
For end-to-end cases when there is a parallel run length > 0 between the two EOL edges with *eolWithin* extension on the checking EOL edge, the `endToEndSpacing` applies.



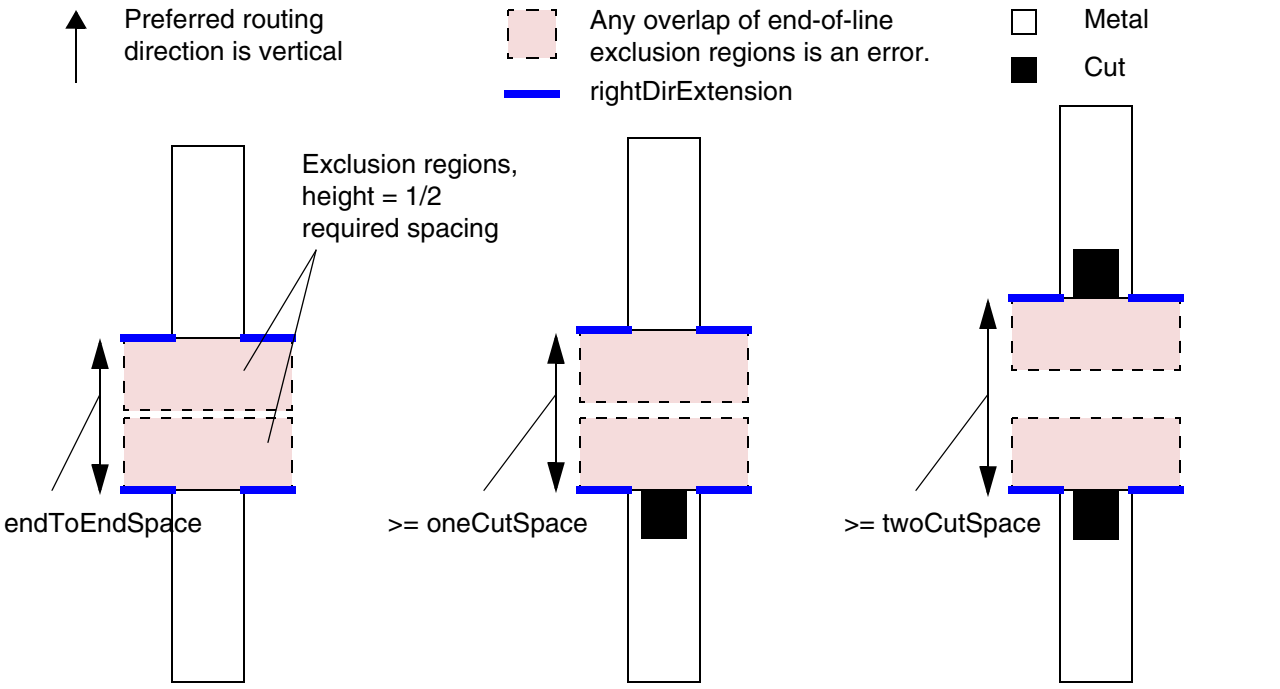
minEndOfLineSpacing with wrongDirWithin



minEndOfLineSpacing with cutEolSpace and enclosureEndWidth



minEndOfLineSpacing with endToEndSpacing and rightDirExtension

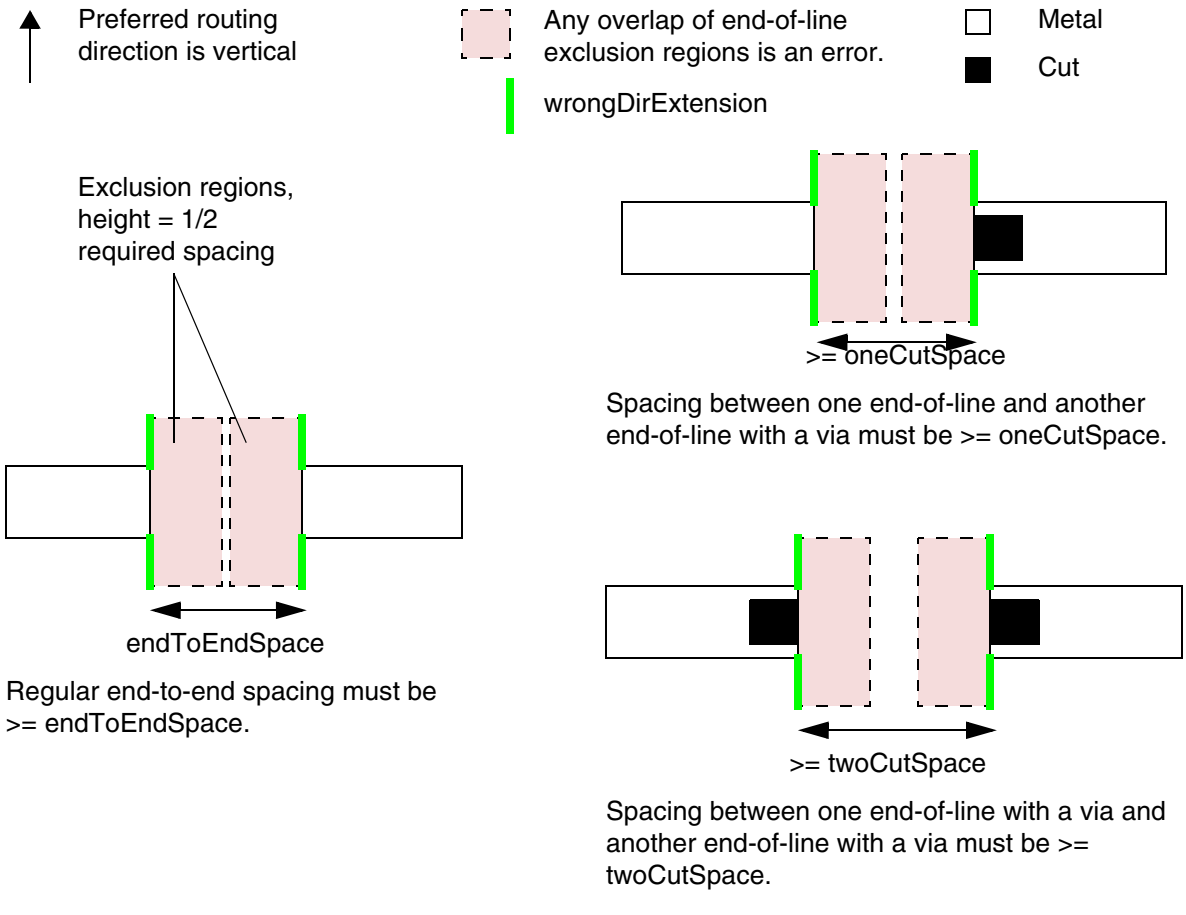


Regular end-to-end spacing must be >= endToEndSpace.

Spacing between one end-of-line and another end-of-line with a via must be >= oneCutSpace.

Spacing between one end-of-line with a via and another end-of-line with a via must be >= twoCutSpace.

minEndOfLineSpacing with endToEndSpacing and wrongDirExtension



Regular end-to-end spacing must be >= endToEndSpace.

Spacing between one end-of-line and another end-of-line with a via must be >= oneCutSpace.

Spacing between one end-of-line with a via and another end-of-line with a via must be >= twoCutSpace.

Parameters

One of the following parameters must be specified:

- `width` (`Value`, required) is the width of the end-of-line wire (*`eolWidth`*). The constraint applies only when the wire width is less than the specified `width` parameter value.
- `widthRangeArray` (`RangeArrayValue`, required) specifies that the constraint applies only to end-of-line edges with edge length in this range, specified in user units.

The following parameters specify the remaining conditions for the constraint:

- `distance` (`Value`, required) is the lateral verification distance (*`eolWithin`*) from the end-of-line wire for the spacing check. The constraint applies anywhere laterally beyond

the end-of-line within (<) `distance`. Also specifies the distance beyond the end-of-line edge for the parallel-edge check region when `count` is greater than 0.

- `count` (`IntValue`, optional) is the number of parallel edges that trigger this constraint and is an integer value of 0, 1, or 2. When this value is 1 or 2, you must also specify `parallelEdgeSpace` and `parallelEdgeWithin` parameters. When count = 2, the two parallel edges must be on opposite sides of the end-of-line.
- `minOppositeWidth` (`Value`, optional) specifies that the constraint applies only if a wire beyond the end-of-line edge has a perpendicular span to the EOL edge that is less than this value.
- `minLength` (`Value`, optional) with the `twoSides` parameter, specifies when the end-of-line applies, based on the side lengths of the end-of-line.
- `twoSides` (`BoolValue`, optional) applies only when `minLength` is given and determines when the constraint applies.

`minLength` with `twoSides false` or not given	The constraint applies when at least one side of the end-of-line is greater than or equal to `minLength`. (The constraint does not apply if both sides of the end-of-line are less than `minLength`.)
`minLength` with `twoSides true`	The constraint applies when both sides of the end-of line are greater than or equal to `minLength`. (The constraint does not apply if either side of the end-of-line is less than `minLength`).

- `maxLength` (`Value`, optional) specifies that the constraint does not apply if both sides of the end-of-line are longer than `maxLength`.
- `endToEndSpacing` (`Value`, optional) specifies that the constraint only applies when there is another end-of-line within (less than) this distance. Use this parameter to specify end-to-end `minEndOfLineSpacing` that is different from `minEndOfLineSpacing` required for end-to-perpendicular shapes.
  - `oneCutSpace` (`Value`, optional) applies only when `endToEndSpacing` is specified. The `oneCutSpace` parameter specifies the spacing required between an end-of-line which touches a single cut, and another that does not.
  - `twoCutSpace` (`Value`, optional) applies only when `endToEndSpacing` is specified. The `twoCutSpace` parameter specifies the spacing required between two end-of-lines, both of which touch a via cut.

- otherEndWidth (Value, optional) specifies that another shape's width must be less than this value for it to be considered an end-of-line. This is used with the endToEndSpacing parameter to qualify the other end-of-line and with the eolAddSpace parameter to extend the end for stricter spacing.
- bothWires (BoolValue, optional) if true, specifies that the endToEndSpacing check only applies if both shapes satisfy the parameters minLength and maxLength. The bothWires parameter should be only specified if endToEndSpacing and at least one of minLength or maxLength is specified, else it is not applicable. The default value of this parameter is false. When false, only one of the two shapes must satisfy the minLength and maxLength parameters.

- equalRectWidth (BoolValue, optional) specifies that the constraint applies only when the length of the end-of-line edge is smaller than or equal to the wire width. If there are more than one minEndOfLineSpacing constraints with the equalRectWidth parameter set for a given layer, then all of them must have equalRectWidth set.
- eolAddSpace (OneDTblValue, custom/optional) specifies that any shape whose width maps to a non-zero spacing value in the table is effectively extended by the given amount. In addition, the spacing measurement becomes Euclidean (radial). By default, Manhattan spacing is used for end-of-line spacing.
- rightDirExtension (Value, optional) specifies the lateral checking distance used to check end-of-line spacing between two end-of-lines routed in the preferred direction.
- wrongDirExtension (Value, optional) specifies the lateral checking distance used to check end-of-line spacing between two end-of-lines routed in the non-preferred direction.
- oaConnectivityType (IntValue, optional) specifies the shapes to which this constraint applies, based on their connectivity, as given in Table on page 734.

oaConnectivityType Parameter Values

String	Value	Constraint applies to:
anyConnectivity	0	(Default) Any connectivity
sameNetConnectivity	1	Same net only
sameIslandConnectivity	2	Contiguous (same metal) shapes only

- oaSpacingDirection (StringAsIntValue, optional) specifies the measurement direction.
  anyDirection 0 Horizontally and vertically (default)

horizontalDirection	1	Horizontally only
verticalDirection	2	Vertically only

- `extendBy` (`Value`, optional) specifies that the end-of-line edge must be extended in the projection direction by this distance before checking. This extension applies to both the end-of-line edge and the adjacent shape.
- `sizeBy` (`Value`, optional) specifies that the end-of-line edge must be extended by this distance on both sides before checking.
- `wrongDirWithin` (`Value`, optional) specifies the lateral checking distance used for wires running in the *non-preferred* direction, which is different from that used for preferred direction wires. If this parameter is specified, the existing `distance` parameter only specifies the *preferred* direction lateral verification distance.
- `diffMask` (`BoolValue`, optional), when `true`, specifies that the constraint applies only between shapes on different masks. The default is `false`.
- `sameMask` (`BoolValue`, optional) specifies whether the constraint applies only between shapes on the same mask (`true`) or between all shapes on the given layer (`false`/ default). When `true`, the constraint specifies end-to-side or end-to-end spacing for advanced nodes. End-to-end same mask spacing also requires `endToEndSpacing` and `otherEndWidth` parameters.
- `cutEolSpace` (`Value`, optional) specifies the spacing required if the metal end-of-line edge under consideration intersects a cut shape on the cut layer either above or below the metal layer. This is spacing to another non-end-of-line edge.
  - `cutClass` (`DualValue`, optional) used with `cutEolSpace` to specify the via width and height for which the constraint applies.
  - `cutLayerEol` (`IntValue`, optional) used with `cutEolSpace` to specify that the cutEolSpace is only applied if the metal end-of-line touches a cut shape in the cut layer above, below, or both (default).
  - `enclosureEndWidth` (`Value`, optional) used with `cutEolSpace`. The `minEndOfLineSpacing` constraint applies to an end-of-line edge touching a via cut if the edge length is less than this parameter value.
  - `enclosureEndWithin` (`Value`, optional) specifies the lateral verification distance for ends-of-lines touching via cuts applied in end-of-line to non-end-of-line spacing. If this parameter is specified, then `enclosureEndWidth` must also be specified. See `rightDirExtension` and `wrongDirExtension` for lateral verification distance used in end-to-end spacing checks.
- `exactEolWidth` (`BoolValue`, optional) specifies that the rule applies only if the edge width is equal to the specified end-of-line width.

- `wrongDirSpace` (`IntValue`, optional) specifies the minimum end-of-line spacing to be used for wires running in the *non-preferred* direction, which is different from that used for preferred direction wires.
- `exceptExactAligned` (`BoolValue`, optional), when `true`, specifies that the constraint does not apply to shapes that are aligned.

These parameters specify conditions when there are edges parallel to the end-of-line. The `count` parameter specifies the number of parallel edges (1 or 2) that must exist for the constraint to apply.

- `parallelEdgeSpace` (`Value`, required when `count` > 0) is the distance (*`parSpace`*) perpendicular to the end-of-line wire.
- `parallelEdgeWithin` (`Value`, required when `count` > 0) is the distance (*`parWithin`*) along the end-of-line wire.
- `parallelEdgeMinLength` (`Value`, optional) specifies that the constraint applies only when the end-of-line length is greater than or equal to this value.
- `subtractEndOfLineWidth` (`BoolValue`, optional) specifies that the *`parSpace`* should be subtracted by the width of the end-of-line edge to define the distance in searching for a parallel neighbor edge.

The following parameters specify that the rule only applies if there is a cut above or below this metal that is less than `enclosedDistance` from the end-of-line edge, and the cut-edge-to-metal-edge space beyond the end-of-line edge is less than `cutToMetalSpacing`. If there is more than one cut connecting the same metal shapes above and below, only one cut needs to meet this rule.

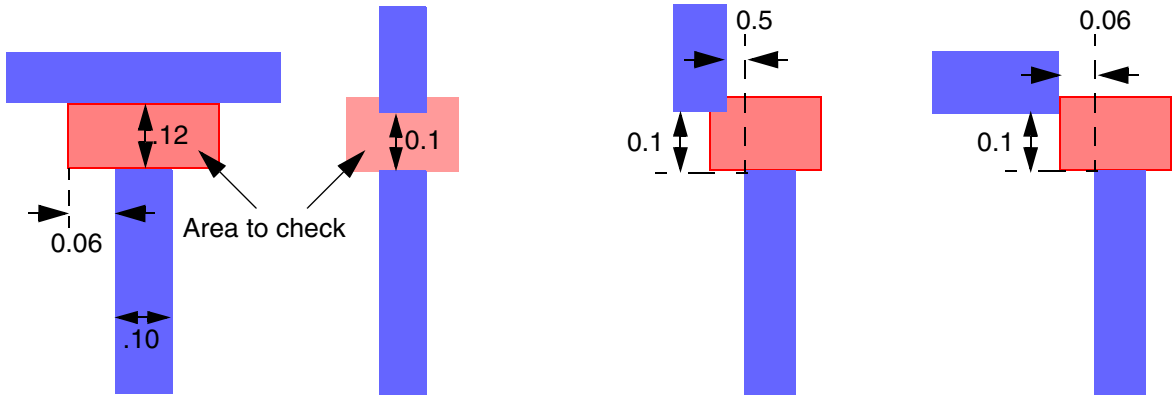
- `cutToMetalSpacing` (`Value`) specifies that the constraint applies only when there is a cut shape (`enclosedCut`) on the layer above or below the metal layer that is less than or equal to this distance from the metal edge.
- `enclosedCut` (`StringAsIntValue`) specifies which cuts are considered with `cutToMetalSpacing` as given in Table on page 472.
- `enclosedDistance` (`Value`, optional) specifies that the rule only applies if there is a cut above or below this metal that is less than `enclosedDistance` from the end-of-line edge, and the cut-edge-to-metal-edge space beyond the end-of-line edge is less than `cutToMetalSpacing`.
- `allCuts` (`BoolValue`, optional) When `true`, the constraint is checked for all cuts connecting the same metal shapes above and below.

Examples

Example 1: minEndOfLineSpacing with count=0

In this example, any end-of-line that is less than 0.11μm wide requires spacing that is greater than or equal to 0.12 μm beyond the end-of-line, within 0.06 μm to either side. The following figure includes examples of this rule.

minEndOfLineSpacing with count=0



a) OK. Has 0.12 μm spacing.

b) Violation; has only 0.10 μm spacing.

c) Violation; has < 0.12μm spacing within 0.06 μm away.

d) OK. Has >= 0.12μm spacing within (<) 0.06μm away.

Format	Example

Tcl	`set_constraint_parameter -name width -Value .11`
	`set_constraint_parameter -name distance -Value 0.06`
	`set_constraint_parameter -name count -IntValue 0`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value .12`

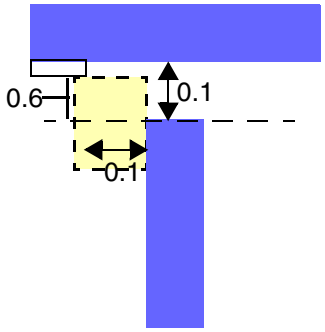
LEF	`PROPERTY LEF58_SPACING \"SPACING 0.12 ENDOFLINE 0.11 WITHIN 0.06 ;\"` `;`

Virtuoso	`spacings(`
	`( minEndOfLineSpacing \"Metal2\" (0.11 0.06 0.0 0.12) )`
	`)`

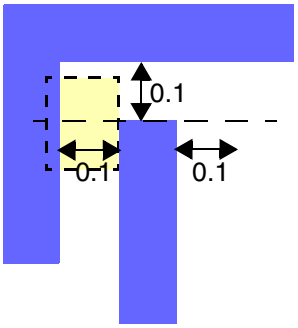
Example 2: minEndOfLineSpacing with count=1

In this example, any end-of-line shape that is less than 0.11 μm wide, with a parallel edge that is less than 0.12 μm away, and is within 0.05 μm of the end-of-line, requires spacing greater than or equal to 0.12μm beyond the end-of-line, when within 0.6 μm to either side of the end-of-line. The following figure includes examples of legal spacing for, and violations of, this rule.

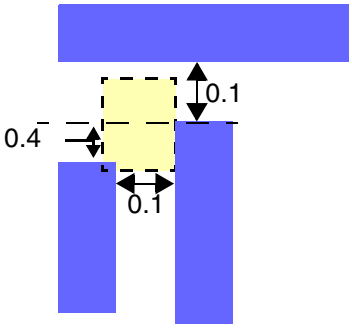
minEndOfLineSpacing with count=1



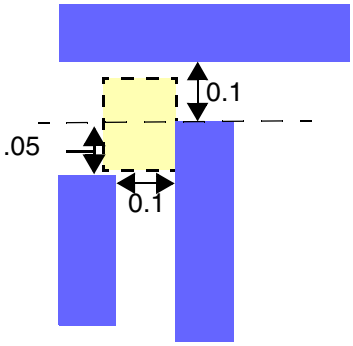
a) OK, no parallel edge.



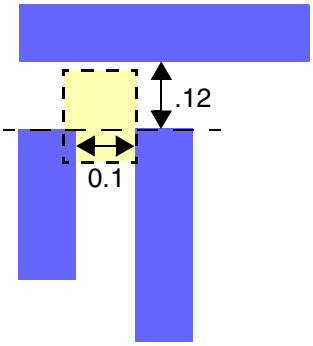
b) Violation. Has parallel edge, needs 0.12 spacing at end-of-line.



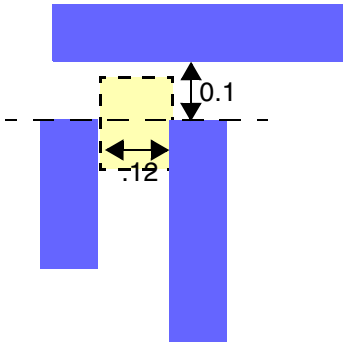
c) Violation. Has parallel edge, needs 0.12 spacing at end-of-line.



d) OK. No parallel edge.



e) OK. Has parallel edge, but has 0.12 spacing at end-of-line.



f) OK. No parallel edge (left edge is >= 0.12 away).

Format	Example

Tcl	`set_constraint_parameter -name width -Value .11`
	`set_constraint_parameter -name distance -Value 0.06`
	`set_constraint_parameter -name count -IntValue 1`
	`set_constraint_parameter -name parallelEdgeSpace -Value .12`
	`set_constraint_parameter -name parallelEdgeWithin -Value .05`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value .12`

LEF	`PROPERTY LEF58_SPACING "SPACING 0.12 ENDOFLINE 0.11 WITHIN 0.06 PARALLELEDGE 0.12 WITHIN 0.05 ;" ;`

Virtuoso	`spacings(`
	`( minEndOfLineSpacing "Metal2" (0.11 0.06 0.001 0.12 50 0.12) )`
	`)`

Example 3: minEndOfLineSpacing with count=2

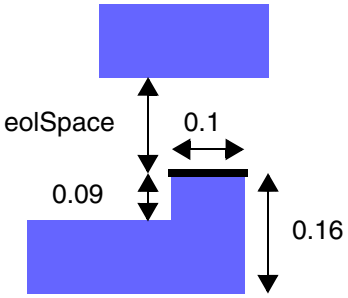
This example is similar to Example 2: minEndOfLineSpacing with count=1, except that the rule is triggered by two parallel edges. See minEndOfLineSpacing with count = 2 for a graphic example.

Format	Example

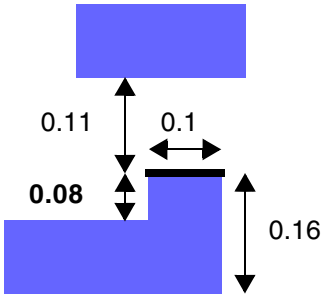
Tcl	`set_constraint_parameter -name width -Value .11`
	`set_constraint_parameter -name distance -Value 0.06`
	`set_constraint_parameter -name count -IntValue 2`
	`set_constraint_parameter -name parallelEdgeSpace -Value .12`
	`set_constraint_parameter -name parallelEdgeWithin -Value .05`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value .12`

LEF	`PROPERTY LEF58_SPACING "SPACING 0.12 ENDOFLINE 0.11 WITHIN 0.06 PARALLELEDGE 0.12 WITHIN 0.05` **`TWOEDGES`** `;" ;`
Virtuoso	`( minEndOfLineSpacing "Metal2" (0.11 0.06 0.002 0.12 .05 0.12) )`

Example 4: minEndOfLineSpacing with maxLength



a) Both sides of the end-of-line are longer than the `maxLength` of 0.08 so the constraint does not apply and eolSpace is not checked.



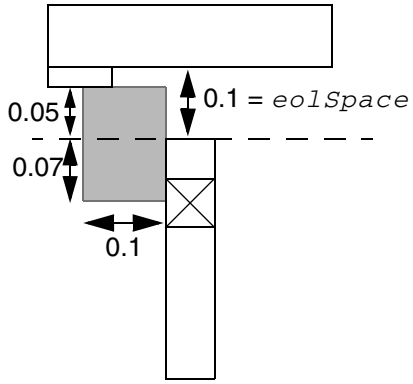
b) Violation. The left side of the end-of-line is shorter than `maxLength` so the constraint applies, and *`eolSpace`* (0.11) < `minEndOfLineSpacing` (0.12).

Format	Example

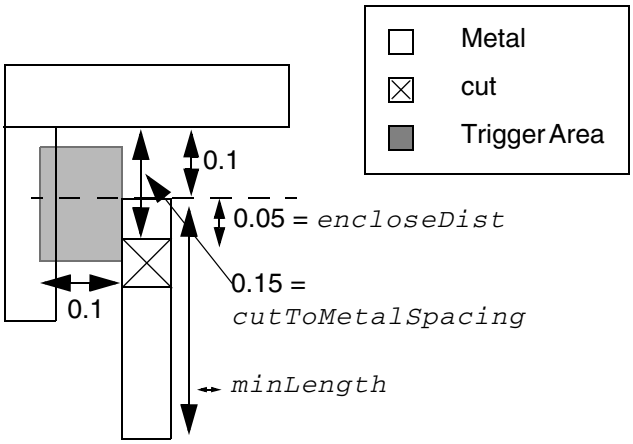
Tcl	`set_constraint_parameter -name width -Value .1`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name maxLength -Value 0.08`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value 0.12`

Format	Example
LEF	`PROPERTY LEF58_SPACING` `"SPACING 0.12 ENDOFLINE 0.1 WITHIN 0.05 MAXLENGTH 0.08 ;" ;`

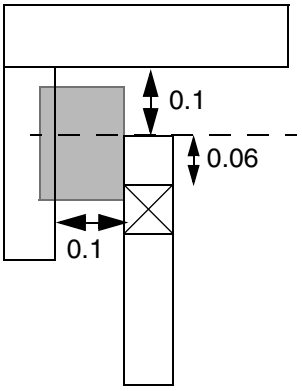
Example 5: minEndOfLineSpacing with parallelEdgeMinLength and cutToMetalSpacing



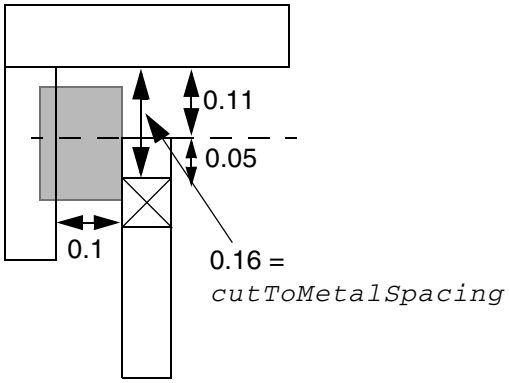
a) No violation. No parallel edge inside trigger area (*parSpace* < 0.10, *eolWithin* < 0.05, and *parWithin* < 0.07), so constraint does not apply.



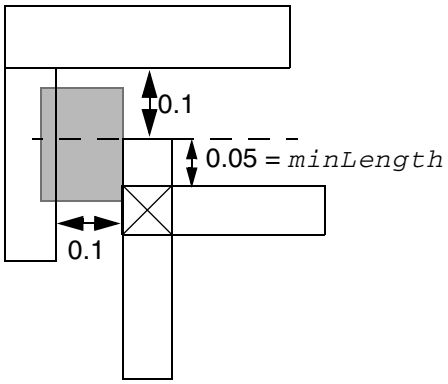
b) Violation. Parallel edge in the trigger area, *minLength* > parallelEdgeMinLength, cut enclosure *encloseDist* < 0.06, *cutToMetalSpacing* < 0.16, so constraint applies but 0.1 < minEndOfLineSpacing of 0.15.



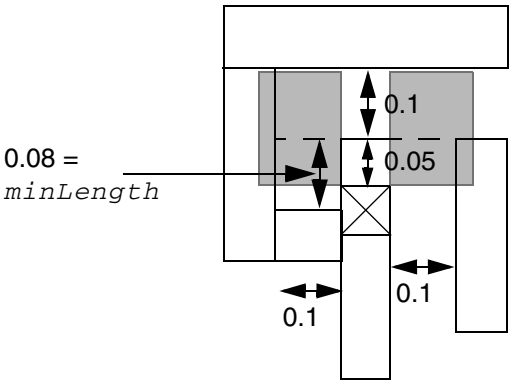
c) No violation. Parallel edge in trigger area, *minLength* > parallelEdgeMinLength, but cut enclosure *encloseDist* >= 0.06, so constraint does not apply.



d) No violation. Parallel edge in the trigger area, *minLength* > parallelEdgeMinLength, cut enclosure *encloseDist* < 0.06, but *cutToMetalSpacing* >= 0.16, so constraint does not apply.



e) No violation. *minLength* < 0.10 parallelEdgeMinLength, so the constraint does not apply.



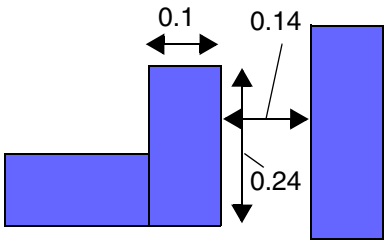
f) Violation. Constraint does not apply to left parallel edge because *minLength* < 0.10, but the right side has a parallel edge, so needs 0.15 *eolSpace* or *cutToMetalSpacing* >= 0.16.

Format	Example

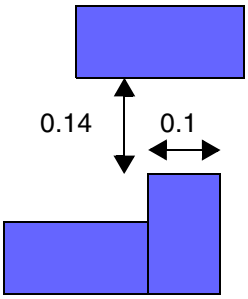
Tcl	`set_constraint_parameter -name width -Value 0.10`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 1`
	`set_constraint_parameter -name parallelEdgeSpace -Value 0.10`
	`set_constraint_parameter -name parallelEdgeWithin -Value 0.07`
	`set_constraint_parameter -name parallelEdgeMinLength -Value 0.10`
	`set_constraint_parameter -name enclosedCut \`
	`  -StrinAsIntValue cutIsBelow`
	`set_constraint_parameter -name enclosedDistance -Value 0.06`
	`set_constraint_parameter -name cutToMetalSpacing -Value 0.16`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`  -Value 0.15`

LEF	`PROPERTY LEF58_SPACING`
	`"SPACING 0.15 ENDOFLINE 0.10 WITHIN 0.05 PARALLELEDGE 0.10`
	` WITHIN 0.07 MINLENGTH 0.10 ENCLOSECUT BELOW 0.06 CUTSPACING 0.16 ;" ;`

Example 6: minEndOfLineSpacing with equalRectWidth



a) OK. The constraint does not apply because the EOL edge of 0.24 is not <= the wire width of 0.1, as required when `equalRectWidth` is specified.



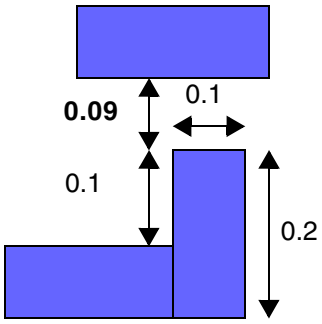
b) Violation. The 0.1 edge has a wire width = 0.1, which satisfies the `equalRectWidth` requirement, so the constraint applies and 0.14 < 0.15 `minEndOfLineSpacing`.

Format	Example

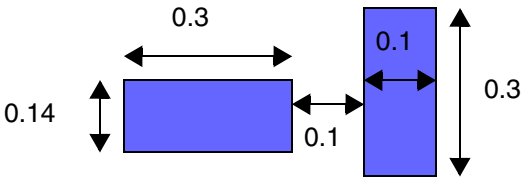
Tcl	`set_constraint_parameter -name width -Value 0.25`
	`set_constraint_parameter -name distance -Value 0.10`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name equalRectWidth -BoolValue true`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value 0.15`

LEF	`PROPERTY LEF58_SPACING` `"SPACING 0.15 ENDOFLINE 0.25 WITHIN 0.10 EQUALRECTWIDTH ;" ;`

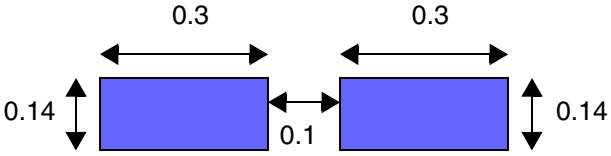
Example 7: minEndOfLineSpacing with endToEndSpacing and minLength



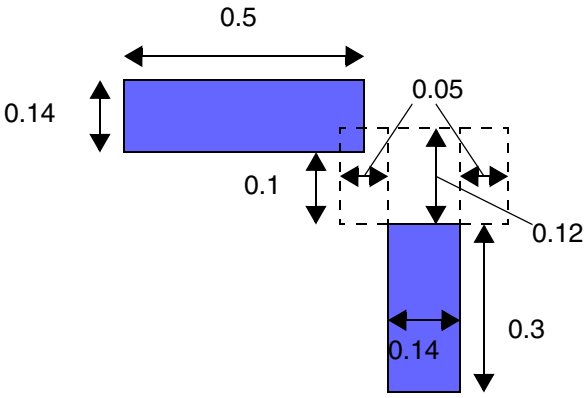
a) Violation. The constraint applies because the right EOL length > 0.11 (`minLength`). EOL spacing must be >= 0.1.



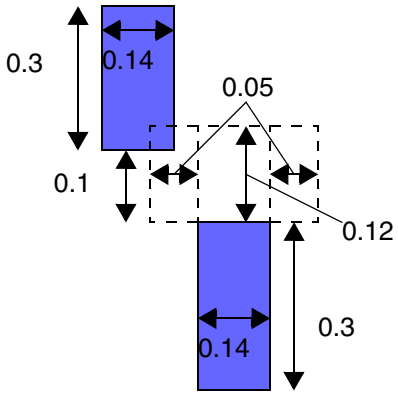
b) OK. The left edge of the right shape is not an EOL edge, so only 0.1 spacing is needed.



c) Violation. End-to-end spacing of 0.12 is needed between the two shapes.



d) OK. Fulfills the end-to-line EOL spacing of 0.1, and the two EOL edges do not have a common parallel run length, so end-to-end spacing is not needed.



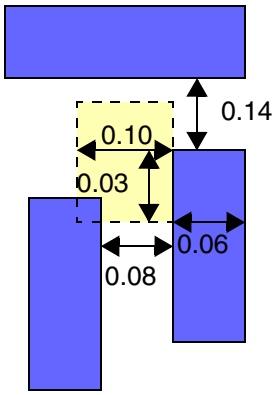
e) Violation. There is a parallel run length > 0 within 0.05 of the EOL edge and 0.1 spacing does not satisfy the end-to-end spacing requirement of 0.12.

Format	Example

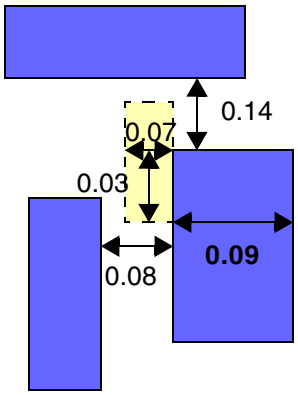
Tcl	`set_constraint_parameter -name width -Value 0.15`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name endToEndSpacing -Value 0.12`
	`set_constraint_parameter -name minLength -Value 0.11`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value 0.10`

LEF	`PROPERTY LEF58_SPACING`
	`"SPACING 0.10 ENDOFLINE 0.15 WITHIN 0.05`
	`ENDTOEND 0.12`
	`MINLENGTH 0.11 ;" ;`

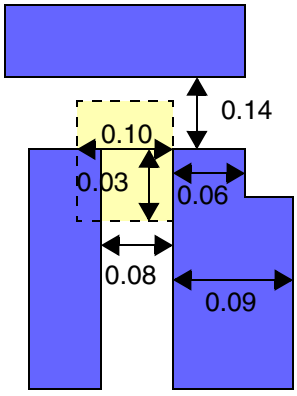
Example 8: minEndOfLineSpacing with subtractEndOfLineWidth



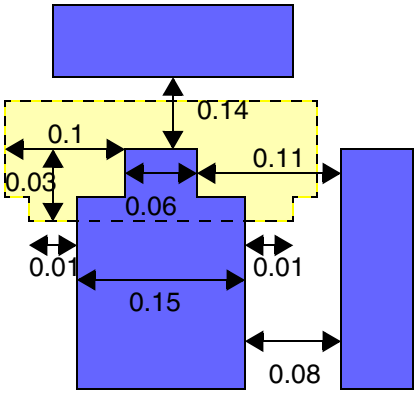
a) Violation. The left shape is < 0.1 (0.16- 0.06) from the EOL wire and triggers the constraint. The horizontal wire must be 0.15 from the EOL to avoid the violation.



b) OK. The constraint does not apply because the parallel edge is outside the trigger area and is > 0.07 (0.16 - 0.09) from the EOL wire.



c) Violation. The left shape is < 0.1 (0.16- 0.06) from the EOL wire and triggers the constraint. The horizontal wire must be 0.15 from the EOL to avoid the violation.



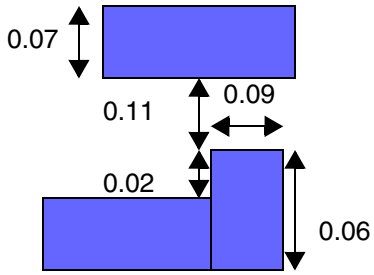
d) OK. The parallel edge is searched by extending the outline of the wire dynamically based on the wire width. Hence, 0.1 (0.16-0.06) on top, and 0.01 (0.16-0.15) on bottom.

Format	Example

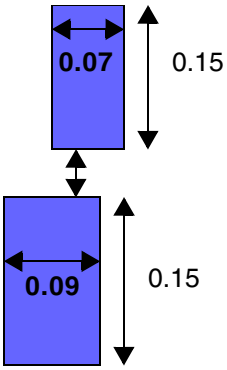
Tcl	`set_constraint_parameter -name width -Value 0.1`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 1`
	`set_constraint_parameter -name subtractEndOfLineWidth \`
	`-BoolValue true`
	`set_constraint_parameter -name parallelEdgeSpace -Value 0.16`
	`set_constraint_parameter -name parallelEdgeWithin -Value 0.03`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value 0.15`

LEF	`PROPERTY LEF58_SPACING`
	`"SPACING 0.15 ENDOFLINE 0.1 WITHIN 0.05`
	`PARALLELEDGE SUBTRACTEOLWIDTH 0.16 WITHIN 0.03 ;" ;`

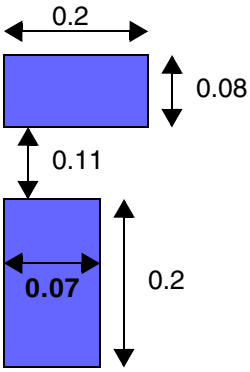
Example 9: minEndOfLineSpacing with minOppositeWidth and minLength



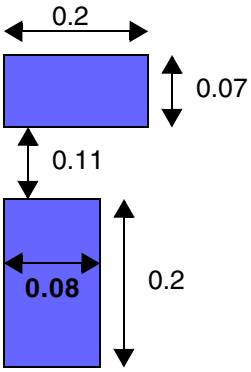
a) Violation. The length on the right side of the EOL >= 0.06 and the opposite wire width < 0.08. Here, 0.11spacing < required 0.12.



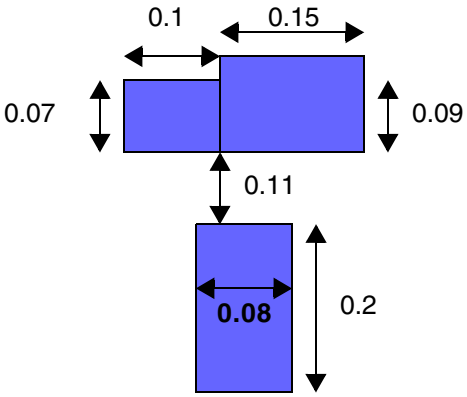
b) OK. The opposite wire has a perpendicular span to the EOL edge of 0.15 >= 0.08 so the constraint does not apply.



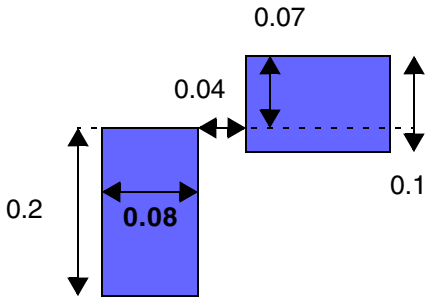
c) OK. Opposite wire width >= 0.08 so the constraint does not apply.



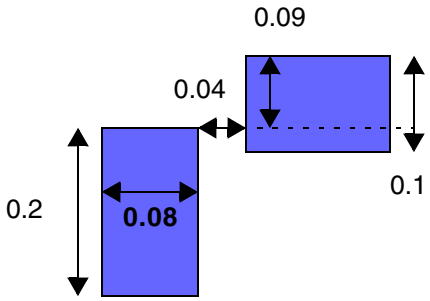
d) Violation. When wire widths from c) are swapped, the opposite wire < 0.08, so the constraint applies.



e) Violation. If any part of the opposite wire width < 0.08, the constraint applies. Here, 0.11 < required 0.12 spacing.



f) Violation. The opposite wire has a perpendicular span to the end-of-line of 0.07 (< 0.08 minOppositeWidth), but is 0.04 from the end-of-line (< 0.05 distance), so the constraint applies and is violated.



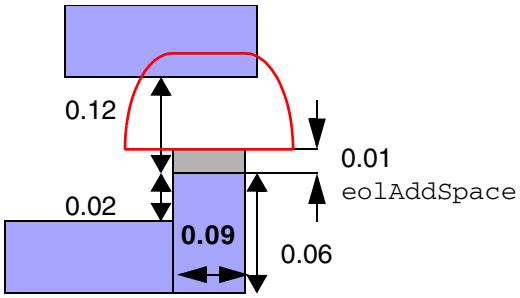
g) OK. The perpendicular span of the opposite wire is 0.09 (>= 0.08 minOppositeWidth), so the constraint does not apply.

Format	Example

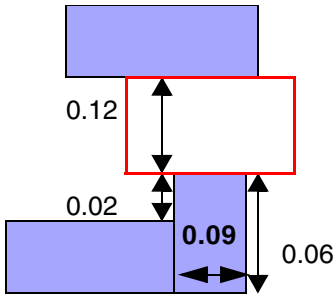
Tcl	`set_constraint_parameter -name width -Value 0.1`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name` **`minOppositeWidth`** `-Value 0.08`
	`set_constraint_parameter -name` **`minLength`** `-Value 0.06`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value 0.12`

LEF	`PROPERTY LEF58_SPACING`
	`"SPACING 0.12 ENDOFLINE 0.1 OPPOSITEWIDTH 0.08 WITHIN 0.05`
	`MINLENGTH 0.06 ;" ;`

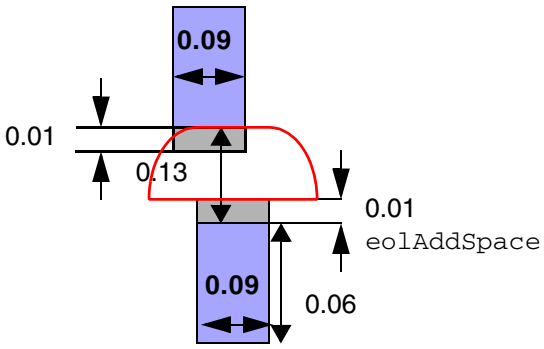
Example 10: minEndOfLineSpacing with eolAddSpace, otherEndWidth and minLength



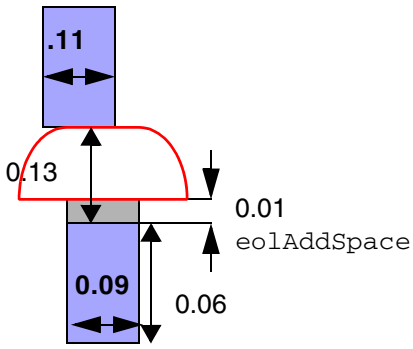
a) Violation. The right side of the EOL >= 0.06 so the constraint applies and the EOL spacing of 0.12 < required 0.12+eolAddSpace. Radial checking is used when `eolAddSpace` is set.



b) OK if `eolAddSpace` is NOT included. In this case, the EOL spacing of 0.12 >= required 0.12. By default, Manhattan checking is performed.



c) Violation. The right side of the EOL >= 0.06 so the constraint applies and the EOL spacing of 0.11 < required due to the extensions on both end-of-lines.

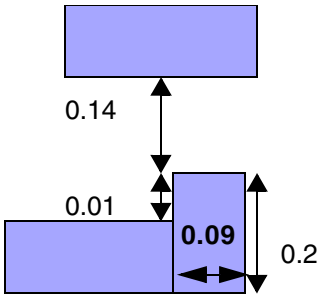


d) OK. Other line end width 0.11 is >= otherEndWidth of 0.1, so the `eolAddSpace` does not apply for that shape.

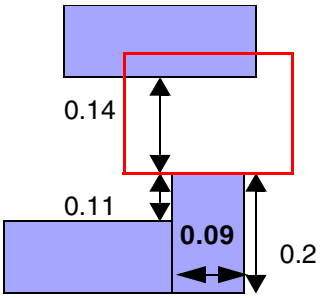
Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.1`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name` **`otherEndWidth`** `-Value 0.1`
	`set_constraint_parameter -name` **`eolAddSpace`** `\`
	`  -OneDTblValue {0 0.01 0.1 0} -row_interpolation snap_down \`
	`  -row_extrapolation { snap_down snap_down)`
	`set_constraint_parameter -name` **`minLength`** `-Value 0.06`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`  -Value 0.12`

Example 11: minEndOfLineSpacing with minLength and twoSides



a) OK. Only one side of the end-of-line is >= `minLength` so the constraint does not apply. If `twoSides` is omitted, the constraint does apply and a violation occurs because the 0.14 spacing < 0.15 `minEndOfLineSpacing`.



b) Violation. Both sides of the end-of-line are >= `minLength` so the constraint applies and 0.14 spacing < 0.15 `minEndOfLineSpacing`.

Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.1`
	`set_constraint_parameter -name distance -Value 0.05`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name` **`minLength`** `-Value 0.11`
	`set_constraint_parameter -name` **`twoSides`** `-BoolValue true`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`-Value 0.15`

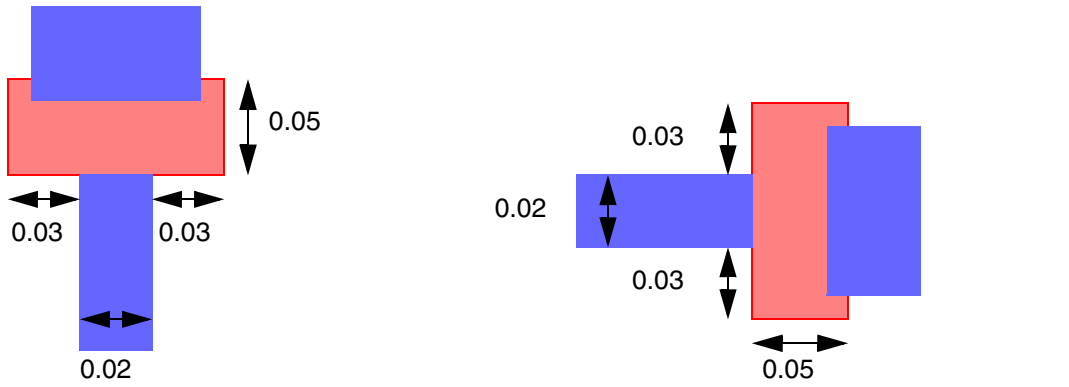
LEF	`SPACING 0.15 ENDOFLINE 0.1 WITHIN 0.05 MINLENGTH 0.11 TWOSIDES`

Example 12: minEndOfLineSpacing with oaSpacingDirection

In this example, the minimum end-of-line spacing measured in the horizontal direction is 0.05 user units for an end-of-line on `Metal2` that is less than 0.03 in width, and applies only to vertical end-of-line edges. The violation area extends 0.03 to the sides of the end-of-line, as shown in the following figure.

Example 12—minEndOfLineSpacing with oaSpacingDirection

```
minEndOfLineSpacing 0.05
width               0.03
distance            0.03
count                  0
oaSpacingDirection  horizontalDirection
```



a) OK. The constraint applies only to spacing measured horizontally from vertical end-of-line edges. This example has a horizontal end-of-line edge so the constraint does not apply.

b) Violation. The end-of-line width is 0.02 (< 0.03 `width`), and the end-of-line edge is vertical, so the constraint applies. The spacing between the end-of-line edge and the other Metal2 shape, measured horizontally, is less than the required minimum end-of-line spacing of 0.05, causing a violation.

Format	Example

Tcl	`set_constraint_parameter -name width -Value 0.03`
	`set_constraint_parameter -name distance -Value 0.03`
	`set_constraint_parameter -name count -IntValue 0`
	`set_constraint_parameter -name oaSpacingDirection \`
	`  -StringAsIntValue horizontalDirection`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`  -Value 0.05`

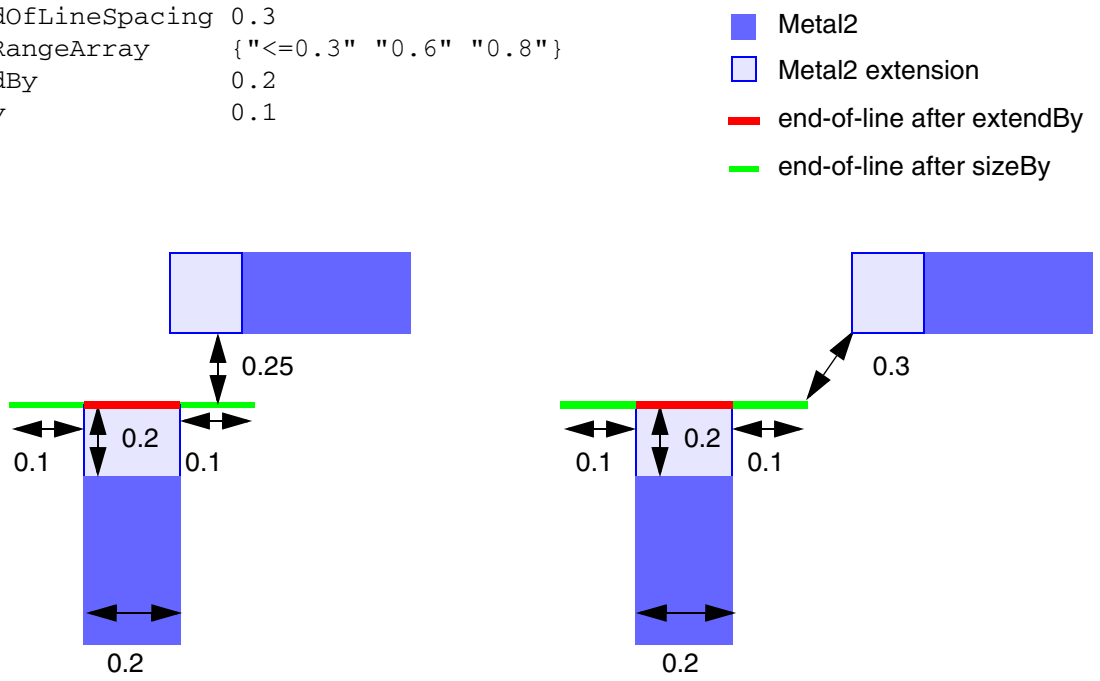
techfile	`spacings(`
	`  ( minEndOfLineSpacing "Metal2"`
	`  'horizontal`
	`  'width 0.03`
	`  'distance 0.03`
	`  0.05`
	`  )`
	`); spacings`

Example 13: minEndOfLineSpacing with widthRangeArray, extendBy and sizeBy

In this example, the minimum spacing is 0.3 user units to an end-of-line edge with width less than or equal to 0.3, or equal to 0.6 or 0.8, when the end-of-line edge and the adjacent shape are extended by 0.2 in the projection direction, and the end-of-line edge is extended by 0.1 on both sides.

Example 13—minEndOfLineSpacing with widthRangeArray, extendBy and sizeBy

```
minEndOfLineSpacing 0.3
widthRangeArray     {"<=0.3" "0.6" "0.8"}
extendBy            0.2
sizeBy              0.1
```



a) Violation. The width of the end-of-line (0.2) is within a specified range (<= 0.3), so the constraint applies but is not met (0.25 < the required 0.3).

b) OK. The width of the end-of-line (0.2) is within a specified range (<= 0.3), so the constraint applies, and it is met.

Format	Example

Tcl	`set_constraint_parameter -name widthRangeArray \`
	`  -RangeArrayValue {"<=0.3" "0.6" "0.8"}`
	`set_constraint_parameter -name extendBy -Value 0.2`
	`set_constraint_parameter -name sizeBy -Value 0.1`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`  -Value 0.3`

Format	Example

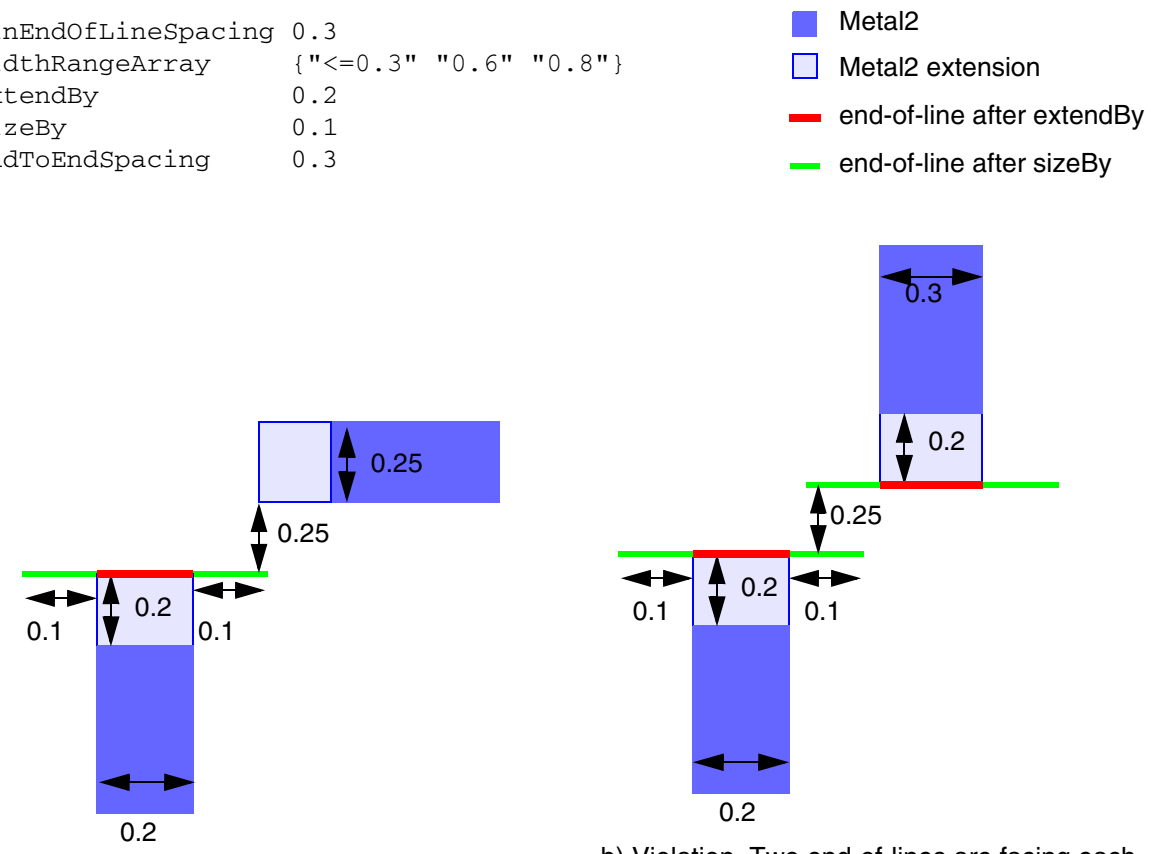
techfile	`spacings(`
	`  ( minEndOfLineSpacing "Metal2"`
	`  'widthRanges ("<=0.3" "0.6" "0.8")`
	`  'extendBy 0.2`
	`  'sizeBy 0.1`
	`  0.3`
	`  )`
	`); spacings`

Example 14: minEndOfLineSpacing with endToEndSpacing, extendBy and sizeBy

In this example, the minimum spacing is 0.3 user units between end-of-line edges with width less than or equal to 0.3, or equal to 0.6 or 0.8, when both end-of-line edges are extended by 0.2 in the projection direction, and the target end-of-line edge is extended by 0.1 on both sides.

Example 14—minEndOfLineSpacing with endToEndSpacing, extendBy and sizeBy

```
minEndOfLineSpacing 0.3
widthRangeArray     {"<=0.3" "0.6" "0.8"}
extendBy            0.2
sizeBy              0.1
endToEndSpacing     0.3
```



a) OK. The constraint applies only to facing end-to-end shapes. These end-of-lines are not facing each other, so the constraint does not apply.

b) Violation. Two end-of-lines are facing each other and are in one of the specified ranges (0.2 and 0.3 are <= 0.3), so the constraint applies but is not met (0.25 < the required 0.3 spacing).

Format	Example

Tcl	`set_constraint_parameter -name widthRangeArray \`
	`  -RangeArrayValue {"<=0.3" "0.6" "0.8"}`
	`set_constraint_parameter -name extendBy -Value 0.2`
	`set_constraint_parameter -name sizeBy -Value 0.1`
	`set_constraint_parameter -name endToEndSpacing -Value 0.3`
	`set_layer_constraint -layer Metal2 -constraint minEndOfLineSpacing \`
	`  -Value 0.0`

Format	Example

techfile	`spacings(`
	`  ( minEndOfLineSpacing "Metal2"`
	`  'widthRanges ("<=0.3" "0.6" "0.8")`
	`  'extendBy 0.2`
	`  'sizeBy 0.1`
	`  'endToEndSpace 0.3`
	`  0.0`
	`  )`
	`); spacings`

Related Topics

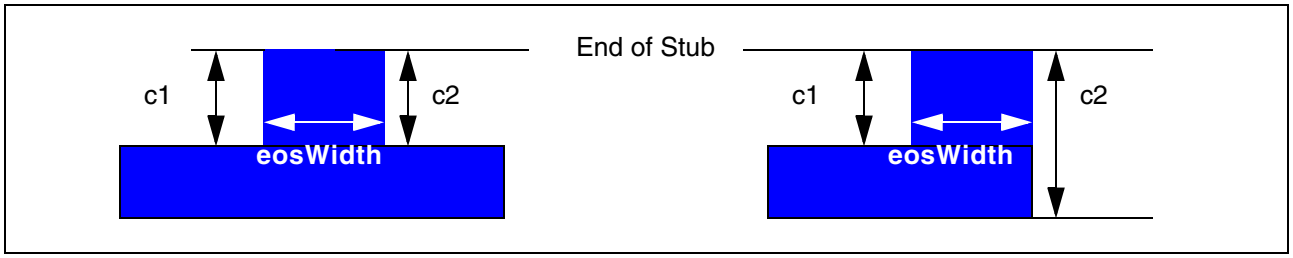
Spacing Constraints

minEndOfStubSpacing

Specifies the minimum spacing between a stub and its neighboring geometry for two types of stubs.

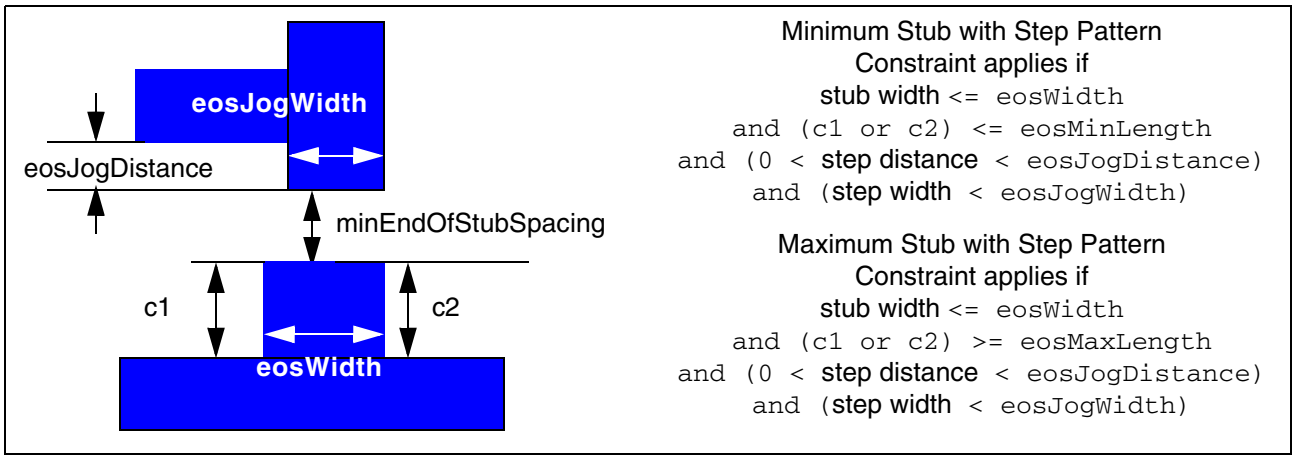
- A *minimum* stub has a side that is less than or equal to a given length (`eosMinLength`).
- A *maximum* stub has a side that is greater than or equal to a given length (`eosMaxLength`).

The constraint applies when any geometry occurs within a region defined by the `minEndOfStubSpacing` constraint, the width of the stub (`eosWidth`), and the side length for maximum stubs (`eosMaxLength`) and/or minimum stubs (`eosMinLength`). The following figure shows examples of stubs where `c1` and `c2` are the stub sides.

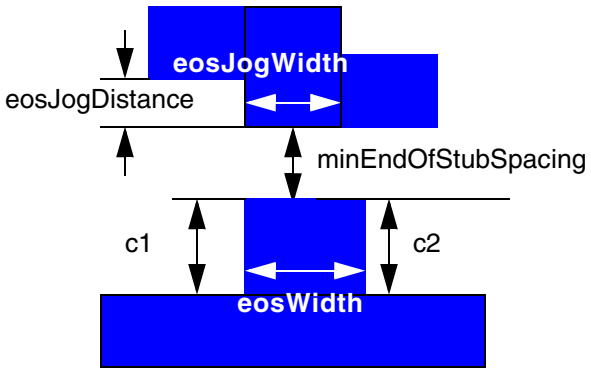


The following types of neighboring geometries will trigger the constraint:

- A *step pattern* with a step distance less than `eosJogDistance` and a step width less than `eosJogWidth`



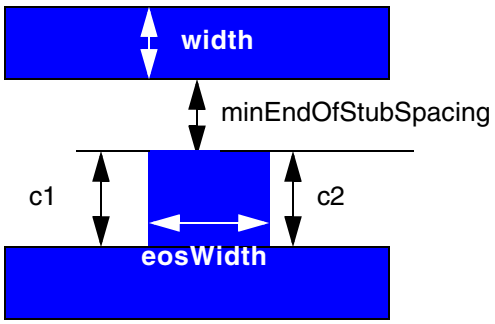
- A *jog pattern* with a jog distance less than `eosJogDistance` and jog width less than `eosJogWidth`



Minimum Stub with Jog Pattern
Constraint applies if
stub width `<= eosWidth`
`and (c1 or c2) <= eosMinLength`
`and (0 <` jog distance `< eosJogDistance)`
`and (`jog width `< eosJogWidth)`

Maximum Stub with Jog Pattern
Constraint applies if
stub width `<= eosWidth`
`and (c1 or c2) >= eosMaxLength`
`and (0 <` jog distance `< eosJogDistance)`
`and (`jog width `< eosJogWidth)`

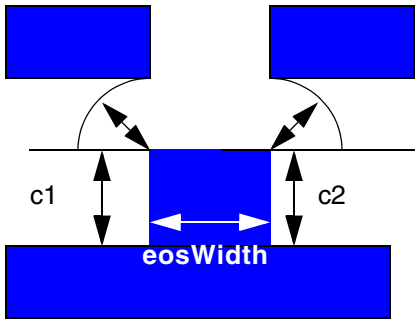
- A perpendicular wire (optionally less than a given `width`)



Minimum Stub with Perpendicular Wire
Constraint applies if
stub width `<= eosWidth`
`and (c1 or c2) <= eosMinLength`
`and (`perpendicular width `< width)`

Maximum Stub with Perpendicular Wire
Constraint applies if
stub width `<= eosWidth`
`and (c1 or c2) >= eosMaxLength`
`and (`perpendicular width `< width)`

- A wire within radial proximity of a stub end corner



Minimum Stub with Radial Extension Constraint applies if stub width `<= eosWidth` `and (c1 or c2) <= eosMinLength` `and (eolRadialCheck` is `true)`

Maximum Stub with Radial Extension Constraint applies if stub width `<= eosWidth` `and (c1 or c2) >= eosMaxLength` `and (eolRadialCheck` is `true)`

minEndOfStubSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

The `minEndOfStubSpacing` constraint has a `Value` that represents the minimum distance required between the end of a stub and its neighboring geometry.

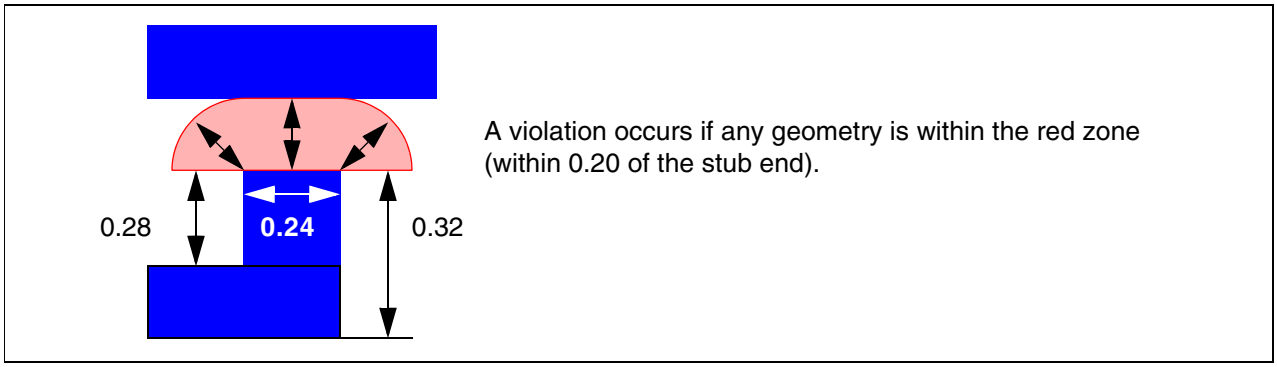
Parameters

- `eosWidth` (`Value`) is the maximum width of the stub. The constraint applies only when the stub width is less than or equal to the specified `eosWidth` parameter value.
- `eosMaxLength` (`Value`) is the minimum length for one side of a maximum stub. The constraint applies only when one side of the stub is greater than or equal to this value.
- `eosMinLength` (`Value`) is the maximum length for one side of a minimum stub. The constraint applies only when one side of the stub is less than or equal to this value.

- `eosJogDistance` (`Value`) and `eosJogWidth` (`Value`) are optional and specify the step pattern parameters. The step pattern is recognized when 0 < step distance < `eosJogDistance` and step width < `eosJogWidth`. A violation occurs if a step pattern is found that is less than `minEndOfStubSpacing` from the stub end.
- `eosJogDistance` (`Value`) and `eosJogWidth` (`Value`) are optional and specify the jog pattern parameters. The jog pattern is recognized when 0 < jog distance < `eosJogDistance` and jog width < `eosJogWidth`. A violation occurs if a jog pattern is found that is less than `minEndOfStubSpacing` from the stub end.
- `width` (`Value`) is optional and specifies the width of metal within the projection of the stub end that will trigger the constraint. This parameter is checked only if neither a step nor jog pattern is detected.
- `eolRadialCheck` (`BoolValue`) is optional and enables radial checking from the stub end corners. This parameter is only checked when `width`, `eosJogDistance`, and `eosJogWidth` are not given.

Examples

The following example for a minimum stub results in a violation whenever any wire is located within `minEndOfStubSpacing` of the stub end.

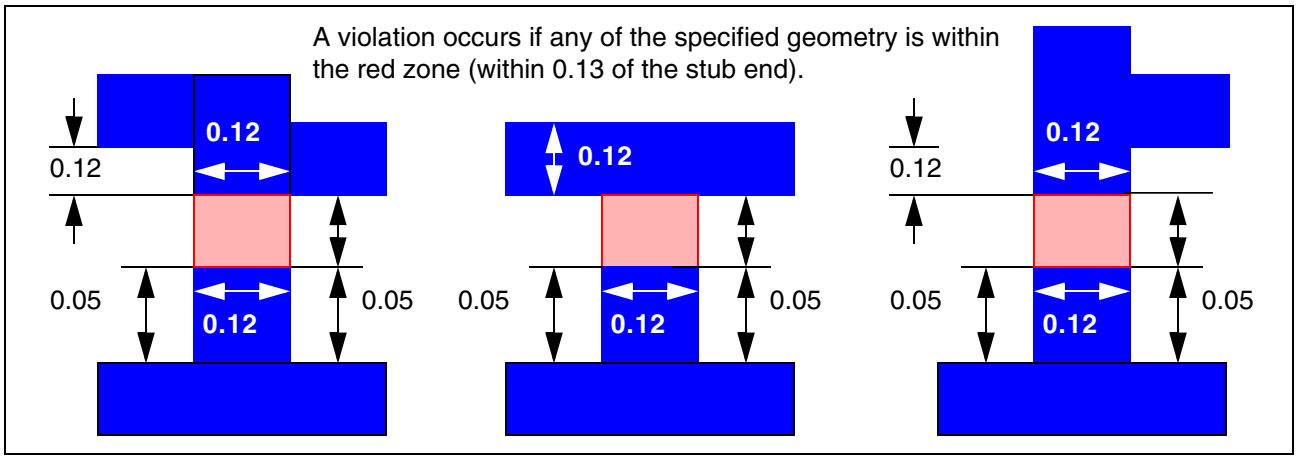



```
set_constraint_parameter -name eosMaxLength -Value 0.28
set_constraint_parameter -name eosWidth -Value 0.24
set_constraint_parameter -name eolRadialCheck -BoolValue true
set_layer_constraint -layer Metal2 -constraint minEndOfStubSpacing -Value 0.20
-hardness hard
```

The following example for a maximum stub results in a violation if one of the following occurs:

- A jog or step pattern is detected that is within 0.13 of the stub end.

- A less than 0.12 wide wire is detected within 0.13 of the stub end.




```
set_constraint_parameter -name eosMaxLength -Value 0.05
set_constraint_parameter -name eosWidth -Value 0.12
set_constraint_parameter -name eosJogDistance -Value 0.12
set_constraint_parameter -name eosJogWidth -Value 0.12
set_constraint_parameter -name width -Value 0.12
set_layer_constraint -layer Metal3 -constraint minEndOfStubSpacing -Value 0.13
-hardness hard
```

Related Topics

Spacing Constraints

minExtensionSpacing

Specifies the minimum spacing for an edge with enclosure that is less than the specified enclosure. Optionally, the constraint only applies for:

- A specified cut class
- An edge containing a long or side edge of a specified cut class.

Spacing Quick Reference

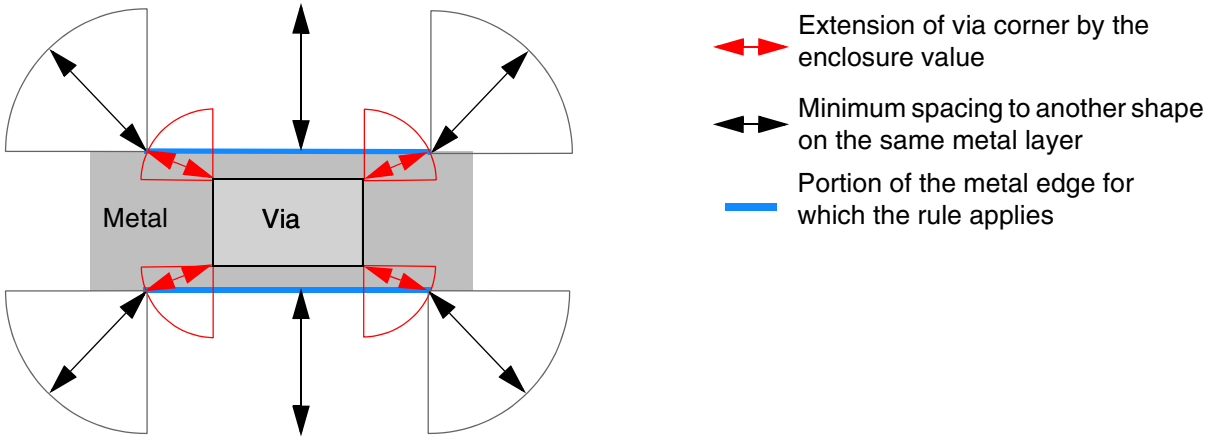
Constraint Type	Layer
Value Types	`OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minExtensionSpacing` constraints have a `OneDTblValue`. The lookup key for the table is the enclosure and the table value is the required spacing. An edge with enclosure less than the given enclosure must have spacing greater than or equal to the given spacing.

If the enclosure is less than the constraint enclosure value, the portion of the metal edge that applies to the constraint is determined by extending the via corner by the enclosure value in Euclidean style to intersect the metal edge, as shown by the blue lines in the following figure.

Determining the Measurement Edge for Spacing



Parameters

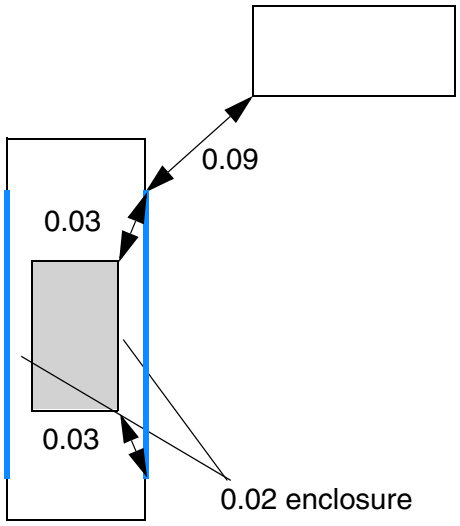
- `layer` (`LayerValue`, required) is the cut layer to which the constraint applies.
- `cutClass` (`DualValue`, optional) specifies the cut class width and length, in user units. Specifies the spacing applies only to an edge with a cut of the given cutClass. If a cut layer has more than one cut class, then this parameter must be specified. You can specify individual rules for each cut class, if required.
- `longEdgeOnly` (`BoolValue`, optional) if set to `true`, specifies that the constraint only applies to a metal edge containing a long/side edge of a rectangular cut with enclosure less than the specified constraint enclosure.

Examples

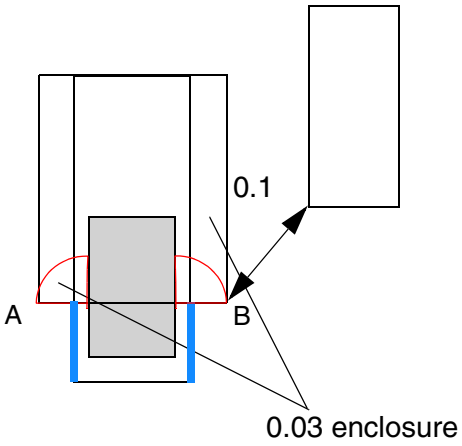
In the following example, `Metal2` edges with enclosure less than 0.03 user units on the long edge of `V2` vias that are 0.06 user units wide and 0.12 user units in length must have spacing greater than or equal to 0.1 user units.

```
set_constraint_parameter -name layer -LayerValue V2
set_constraint_parameter -name cutClass -DualValue {0.06 0.12}
set_constraint_parameter -name longEdgeOnly -BoolValue true
set_layer_constraint -layer Metal2 -constraint Spacing \
 -OneDTblValue {0.03 0.1}
```

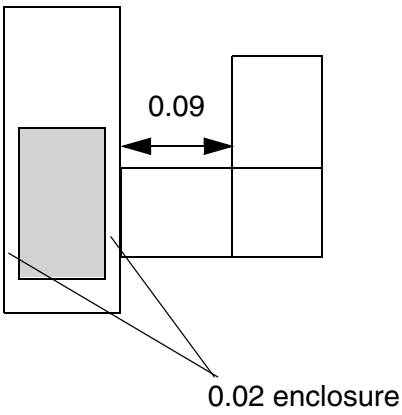
Illustration of `Spacing`



a) Violation, the cut corners are extended by 0.03 to form the blue edges, where the spacing of 0.1 is required and not met.



b) OK, the 0.1 metal spacing only applies to the portion of the cut with enclosure < 0.03. When the upper “corners” of the via are extended, they cross the metal edge at A and B, so the constraint spacing applies to those points and the portion of the edges in blue.



c) Violation, the 0.1 spacing applies to the same-metal jog.

Related Topics

Spacing Constraints

minInnerVertexSpacing

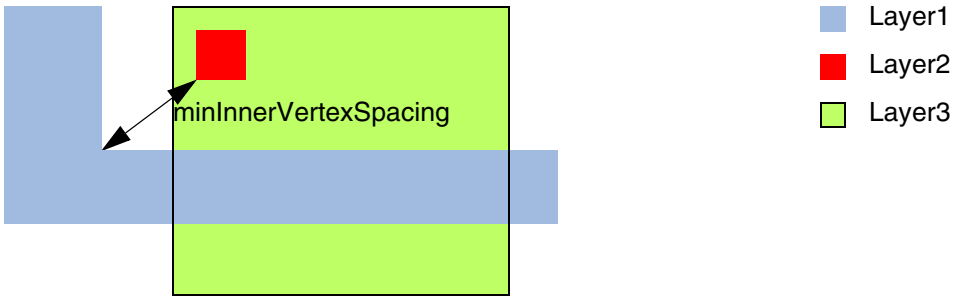
Specifies the minimum spacing between a layer1 inner vertex (concave corner) and a layer2 shape, provided both the layer1 and layer2 shapes overlap a common layer3 shape.

minInnerVertexSpacing Quick Reference

Constraint Type	Layer array
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minInnerVertexSpacing` constraints have a `Value` that represents the minimum spacing, in user units.



Examples

The following example sets the minimum spacing between `Metal1` shapes forming an inner vertex (concave corner) and a Via1 cut shape such that there is a `Metal2` shape that overlaps both the Via1 shape and the `Metal1` shape(s).

Format	Example
Tcl	`set_layerarray_constraint -constraint minInnerVertexSpacing \ -layer_array {Metal1 Via1 Metal2} -Value 0.6`

Format	Example

Virtuoso	`orderedSpacings(`
	`  (minInnerVertexSpacing "Metal1" "Via1" "Metal2"  0.6 )`
	`) ;orderedSpacings`

Related Topics

Spacing Constraints

minJointCornerSpacing

Specifies the minimum spacing required between the corners of two different joints that are facing each other and where there is no parallel run length between the joints.

A joint corner is defined as a convex corner that consists of two consecutive joints with a span length greater than the specified `spanLength`, and which must not be an end-of-line edge of width less than the given `jointWidth`. You can optionally specify that the joint corner has a length greater than or equal to a specified `minLength` along both sides and that the spacing applies only if the edge length of both joints is less than or equal to a specified edge `length`.

minJointCornerSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`Value`	Specifies in user units the minimum spacing required between two facing joint corners.

Required Parameters

`jointWidth`	Specifies the minimum width for the joint. Only joints that are not an end-of-line joint with width greater than this value are considered. Type: `Value`
`spanLength`	Specifies the minimum span length for the joint. Type: `Value`

Optional Parameters

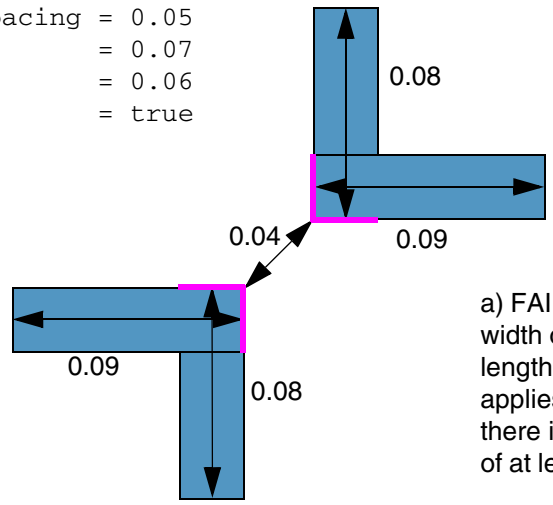
`minLength`	Specifies that minimum length of both sides required for the constraint to apply. Type: `Value`
`length`	Specifies the maximum value for the edge length that forms the joint corner. Type: `Value`
`sameMask`	Specifies whether the spacing applies only to objects on the same mask. The default is `false`. Type: `BoolValue`

Examples

```
set_constraint_parameter -name jointWidth -Value 0.07
set_constraint_parameter -name spanLength -Value 0.06
set_constraint_parameter -name sameMask -BoolValue true
set_layer_constraint -layer Metal1 -constraint minJointCornerSpacing \
  -Value 0.05
```

Specifies that the spacing between joint corners must be greater than or equal to 0.05 user units and the width and span length of a joint must be greater than 0.07 and 0.06 user units respectively. Additionally spacing is only applied to same-mask objects.

```
minJointcornerSpacing = 0.05
  jointWidth          = 0.07
  spanLength          = 0.06
  sameMask            = true
```



a) FAIL. Both joints satisfy the required joint width of 0.07and span of 0.06 and the edge length is less than 0.1 so the constraint applies. However, there is no parallel and there is no parallel run length so a spacing of at least 0.05 is required between the joint

Related Topics

Spacing Constraints

minNotchSpanSpacing

Specifies a spacing between neighbor wires and a notch where the span of the legs on either side is less than `notchSpan`, and the legs are less than `notchSpacing` apart. The spacing applies to the portion of the neighbor wire which has a common parallel run length greater than 0 with the notch legs. In addition, even if the neighbor wires are farther away than the specified spacing, if there are two consecutive U-shaped notches where the span is less than `notchSpan`, and the legs are less than `notchSpacing` apart, it is considered a violation. However, if the length of the smallest notch is `exceptNotchLength`, and the outer edges of the notches has neighbor wires greater than or equal to `minNotchSpanSpacing` away, the rule does not apply.

minNotchSpanSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minNotchSpanSpacing` constraints have a `Value` that represents the spacing, in user units, between a notch leg and a neighboring wire.

Parameters

- `notchSpan` (`Value`, required) The constraint applies when the span of the legs on either side of the notch is less than this value, in user units.
- `notchSpacing` (`Value`, required) specifies the spacing, in user units, between the legs. The constraint applies when the distance between the legs is less than this value, in user units.
- `exceptNotchLength` (`Value`, optional) specifies the exact notch length, in user units, at which this constraint does not apply, as long as neighbor wires are far enough away.

Examples

In this example, the minimum spacing is 0.1 μm between neighbor wires and a notch where the span of the legs on either side is less than 0.15 μm, and the legs are less than 0.12 μm apart. In addition, even if the neighbor wires are farther away than the specified spacing, if there are two consecutive U-shaped notches where the span is less than 0.15 μm, and the legs are less than 0.12 μm apart, it is considered a violation. However, if the length of the smallest notch is 0.13 μm and the distance between the outer edges of the notches and neighbor wires is greater than or equal to 0.1 μm, the rule does not apply.

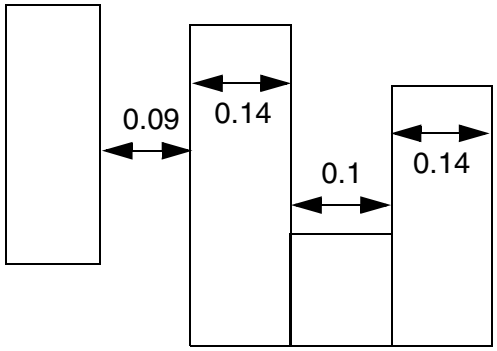
Format	Example

Tcl	`set_layer_constraint -constraint minSpacing -layer Metal1 -Value 0.08`
	
	`set_constraint_parameter -name notchSpan -Value 0.15`
	`set_constraint_parameter -name notchSpacing -Value 0.12`
	`set_constraint_parameter -name exceptNotchLength -Value 0.13`
	`set_layer_constraint -constraint minNotchSpanSpacing \`
	`  -layer Metal1 -Value 0.1`

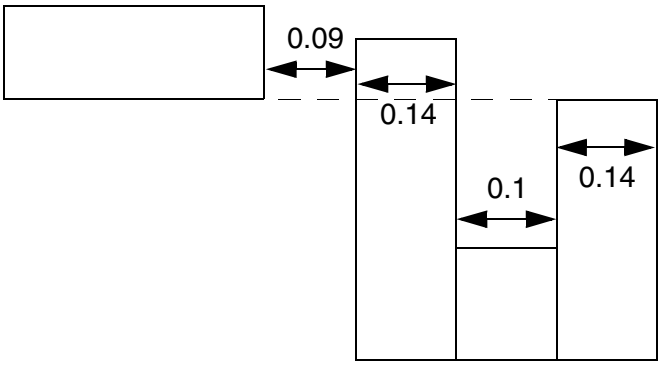
LEF	`SPACING 0.08 ;`
	
	`PROPERTY LEF58_SPACING`
	`"SPACING 0.1 NOTCHSPAN 0.15 NOTCHSPACING 0.12`
	`   EXCEPTNOTCHLENGTH 0.13 ; " ;`

Virtuoso	`spacings( (minSpacing "Metal1" 0.08) ) ;spacings`
	
	`spacings(`
	`( minNotchSpanSpacing Metal1`
	`  'notchSpan 0.15`
	`  'notchSpace 0.12`
	`  'exceptNotchLength 0.13`
	`  0.1 )`
	`); spacings`

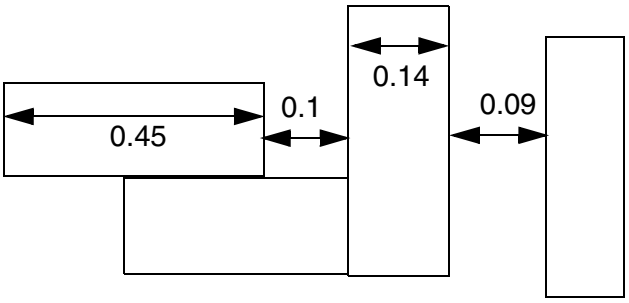
Illustration of minNotchSpanSpacing Example



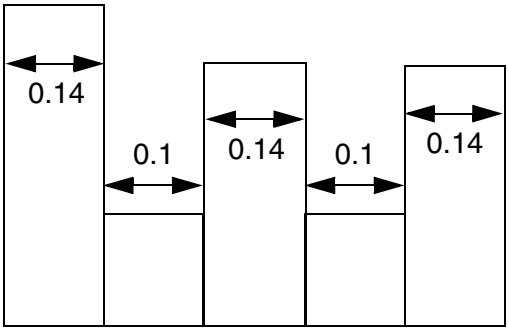
a) Violation, the spacing to the neighbor wire is 0.09 (< required 0.1) for the notch span of 0.14 (< 0.15) and notch spacing of 0.1 (< 0.12).



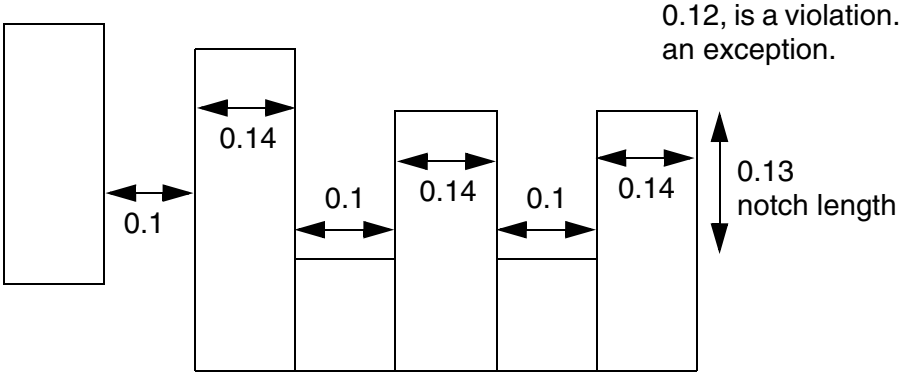
b) OK, the neighbor wire does not have common parallel run length to the common portion of the notch length edge.



c) OK. The left wire has span of 0.45 (>= 0.15), so the rule does not apply.



d) Violation. Multiple consecutive notches, with span length < 0.15 and notch spacing < 0.12, is a violation. See e) for an example of an exception.



e) OK, the length of the shortest notch is equal to `exceptNotchLength` and the distance to the neighbor wire is 0.1 (>= `minNotchSpanSpacing`), so the rule does not apply.

Related Topics

Spacing Constraints

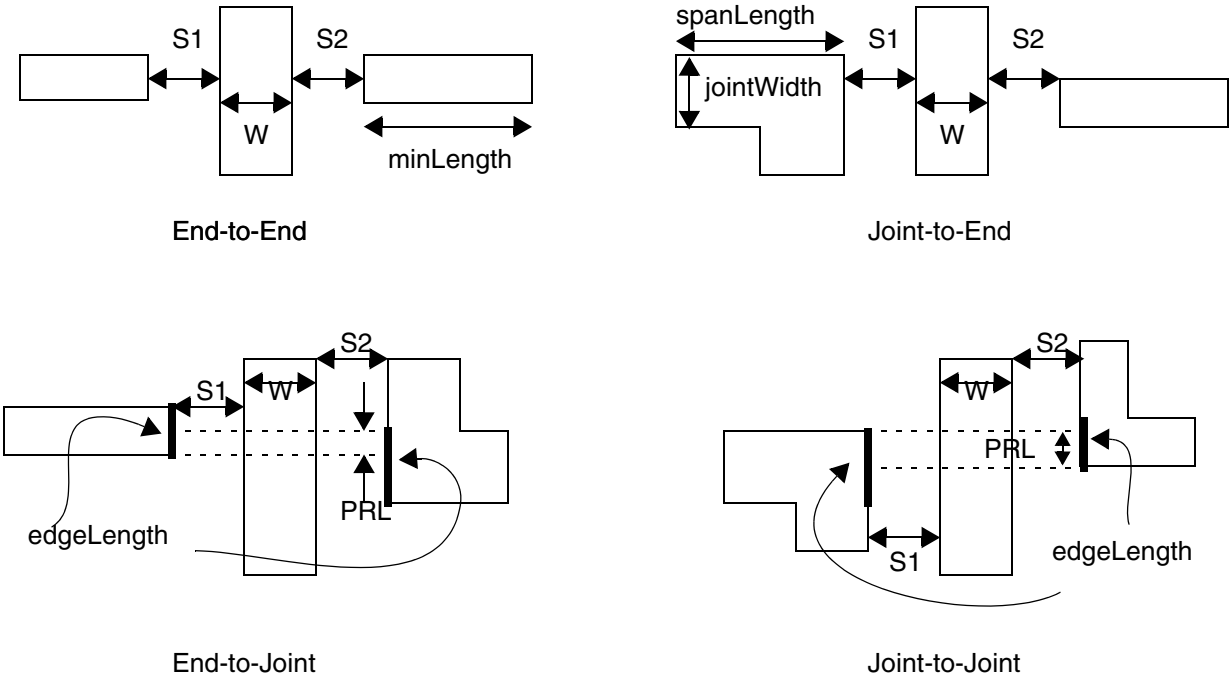
check_space

minOppositeSpanSpacing

Specifies the spacing for a wire that has two neighboring shapes on opposite edges with a projected parallel run length greater than zero. Only neighbor wires that are classified as either *ends* or a *joints* are considered and the constraint applies only when the middle wire width is less than a specified width.

The constraint specifies spacing for each of the four neighbor categories (End-to-End, End-to-Joint, Joint-to-End, and Joint-to-Joint) shown in the following figure.

Neighbor Categories for minOppositeSpanSpacing



minOppositeSpanSpacing Quick Reference

Constraint Type	Layer
Value Types	DualValueTbl
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minOppositeSpanSpacing` constraints have a DualValueTbl that must include a DualValue for each of the four categories in the following table and must be specified in the same order as shown in the table.

Categories for minOppositeSpanSpacing

Category	DualValue (S1 S2)
End-to-End	{endSpacing endSpacing}
End-to-Joint	{endSpacing jointSpacing}
Joint-to-End	{jointSpacing endSpacing}
Joint-to-Joint	{jointSpacing jointSpacing}

- *endSpacing* specifies the minimum spacing between an end neighbor edge to the middle wire.
- *jointSpacing* specifies the minimum spacing between a joint neighbor edge to the middle wire.

To satisfy the rule,

- For End-to-End and Joint-to-Joint cases, either both neighbor spacings must be greater than or equal to the minimum of the specified spacings, or at least one neighbor spacing must be greater than or equal to the maximum of the specified spacings in End-to-End or Joint-to-Joint.
- For End-to-Joint and Joint-to-End cases, both End-to-Joint and Joint-to-End spacings must be fulfilled individually. To fulfill one statement, either the spacing between the middle wire and the joint must be greater than or equal to *joint spacing*, or the spacing between the middle wire and the end must be greater than or equal to *end spacing*.

The following table shows the syntax for setting the `minOppositeSpanSpacing` constraint.

Syntax for minOppositeSpanSpacing

Format	Example

Tcl	`set_constraint_parameter -name width -Value f_width`
	`set_constraint_parameter -name endOfLineWidth -Value f_eolWidth`
	`set_constraint_parameter -name spanlength -Value f_spanLength`
	`[set_constraint_parameter -name length -Value f_minLength]`
	`[set_constraint_parameter -name jointWidth -Value f_jointWidth]`
	`[set_constraint_parameter -name exceptEdgeLength -Value f_edge]`
	`[set_constraint_parameter -name maxlength -Value f_maxPRL]`
	
	`set_layer_constraint -constraint minOppositeSpanSpacing \`
	`  -layer s_layerName -DualValueTbl { \`
	`  f_spacing1_endToEnd  f_spacing2_endToEnd \`
	`  f_spacing1_endToJoint  f_spacing2_endToJoint \`
	`  f_spacing1_jointToEnd  f_spacing2_jointToEnd \`
	`  f_spacing1_jointToJoint f_spacing2_jointToJoint}`

LEF	`PROPERTY LEF58_OPPOSITEEOLSPACING`
	`"OPPOSITEEOLSPACING WIDTH f_width`
	`ENDWIDTH f_eolWidth [MINLENGTH f_minLength]`
	`[JOINTWIDTH f_jointWidth] JOINTLENGTH f_spanLength`
	`  [EXCEPTEDGELENGTH f_edge [PRL f_maxPRL]]`
	`  ENDTOEND  f_spacing1_endToEnd  f_spacing2_endToEnd`
	`  ENDTOJOINT  f_spacing1_endToJoint  f_spacing2_endToJoint`
	`  JOINTTOEND  f_spacing1_jointToEnd  f_spacing2_jointToEnd`
	`  JOINTTOJOINT f_spacing1_jointToJoint f_spacing2_jointToJoint`
	`;" ;`

Parameters

The following parameters are used to determine which neighboring shapes represent ends and joints and when the constraint applies.

- `endOfLineWidth` (`Value`) The constraint applies only when a neighbor end-of-line width (*eolWidth*) is less than this value. If the `jointWidth` parameter is not specified, the `endOfLineWidth` value is also used for `jointWidth`. *eolWidth* measurements are illustrated in Illustration of eolWidth and minLength for End-to-End Spacing.
- `spanLength` (`Value`) The neighbor wire end edge is a joint (and not an end) if it has *eolWidth* less than `endOfLineWidth` and span (*spanLength*) greater than this value. Joints can be either a T or an L pattern. *spanLength* measurements are illustrated in Illustration of jointWidth, spanLength and jointToEdgeEndLength for Joint Spacing.
- `width` (`Value`) The constraint applies only when the middle wire is less than this value.

- `jointWidth` (`Value`, optional) The constraint applies only when a neighbor joint width is less than this value. If this parameter is not specified for the constraint, the `endOfLineWidth` value is used for `jointWidth`.
- `jointToEdgeEndLength` (`Value`, optional) The constraint applies only when the distance (*`jointToEdgeEndLength`*) from the end point of the joint to the end of the edge that contains the joint is greater than or equal to this distance.
- `exceptEdgeLength` (`Value`, optional) The constraint does not apply if both neighbor edges have a length (*`edgeLength`*) greater than or equal to `exceptEdgeLength` and, optionally, a projected parallel run length (*`PRL`*) less than or equal to `maxLength`. Neighbors can be ends or joints.
- `exceptEdgeLengthPRL` (`ValueArrayValue`, optional) The `ValueArrayValue` is mapped to a list of `DualValue` corresponding to `exceptEdgeLength` and `maxLength` or to a `Value` corresponding to just `exceptEdgeLength`. The `ValueArrayValue` can contain up to two entries, as shown below:

  ```
  set_constraint_parameter -name exceptEdgeLengthPRL \
  -ValueArrayValue { -DualIntValue { 0.072 0.01 } }
  ```

  The constraint does not apply if both neighbor edges have a length (*`edgeLength`*) greater than or equal to `exceptEdgeLength` and, optionally, a projected parallel run length (*`PRL`*) less than or equal to `maxLength`. Neighbors can be ends or joints.
- `exceptDualLength` (`Value`, optional) The constraint does not apply if both neighbor edges have a length (*edgeLength*) greater than and or equal to `exceptDualLength`. Neighbors can be ends or joints.
- `length` (`Value`, optional) The constraint applies only when the end-of-line length (*`minLength`*) is greater than or equal to this value along both sides. *`minLength`* measurements are illustrated in Illustration of eolWidth and minLength for End-to-End Spacing.
- `maxLength` (`Value`, optional) The constraint does not apply if both the end or joint neighbor edges have a length (*`edgeLength`*) greater than or equal to `exceptEdgeLength` and a projected parallel run length (*`PRL`*) less than or equal to `maxLength`.

Illustration of eolWidth and minLength for End-to-End Spacing

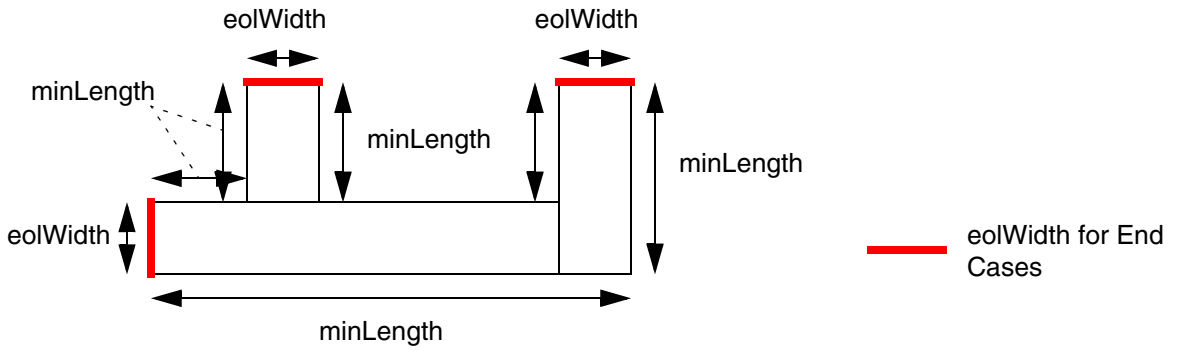
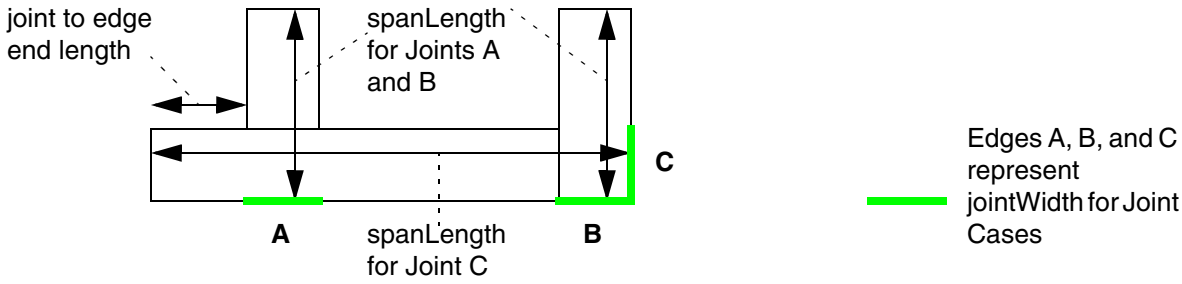


Illustration of jointWidth, spanLength and jointToEdgeEndLength for Joint Spacing



Examples

Format

Example

Tcl

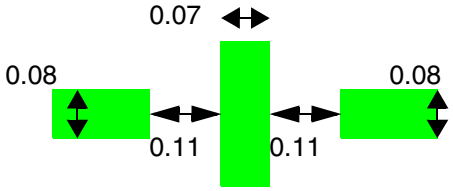
```
set_constraint_parameter -name width            -Value 0.08
set_constraint_parameter -name endOfLineWidth   -Value 0.1
set_constraint_parameter -name spanlength       -Value 0.15
set_constraint_parameter -name exceptEdgeLength -Value 0.08
set_constraint_parameter -name maxlength        -Value 0.03
set_layer_constraint -constraint minOppositeSpanSpacing \
  -layer Metal1 -DualValueTbl \
   { 0.11 0.13 \
     0.12 0.10 \
     0.12 0.10 \
     0.12 0.16 } \
   -row_interpolation snap_down \
   -row_extrapolation {snap_down snap_down}
```

LEF

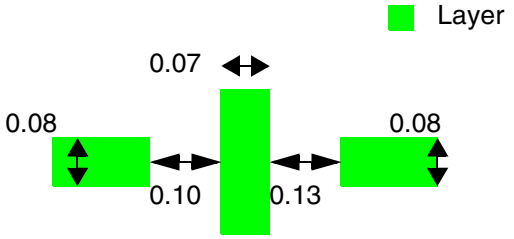
```
PROPERTY LEF58_OPPOSITEEOLSPACING
  "OPPOSITEEOLSPACING WIDTH 0.08 ENDWIDTH 0.1 JOINTLENGTH 0.15
  EXCEPTEDGELENGTH 0.08 PRL 0.03
  ENDTOEND     0.11 0.13
  ENDTOJOINT   0.12 0.10
  JOINTTOEND   0.12 0.10
  JOINTTOJOINT 0.12 0.16 ; " ;
```

The following figure shows how the constraint is interpreted for this example.

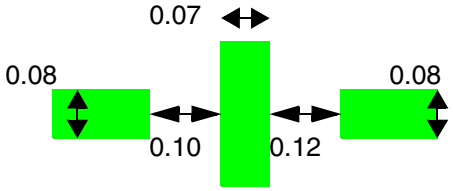
Examples for minOppositeSpanSpacing



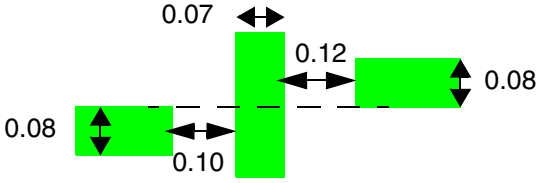
a) OK. Constraint satisfied for end-to-end spacing with spacing to both ends >= 0.11.



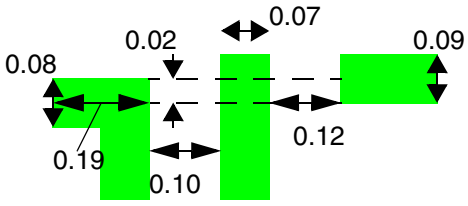
b) OK. Constraint satisfied for end-to-end spacing with one end spacing >= 0.13.



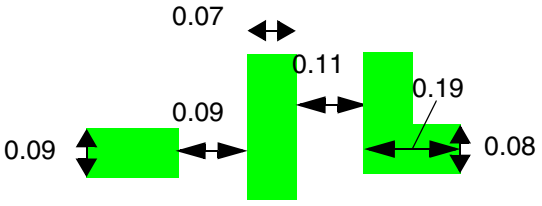
c) Violation. Left spacing (0.10) < 0.11 minimum required end-to-end spacing.



d) OK. Constraint does not apply because there is no projected parallel run length.



e) OK. Constraint does not apply because both neighbor edges >= 0.08 exceptEdgeLength and parallel run length 0.02 <= 0.03 maxLength.



f) Violation.
For end-to-joint {0.12 0.10}, end spacing fails (0.09<0.12), but joint spacing is OK (0.11 > 0.10).
For joint-to-end {0.12 0.10}, both joint spacing (0.11 < 0.12), and end spacing (0.09< 0.10) fail, so the constraint fails.

Related Topics

Spacing Constraints

minOuterVertexSpacing

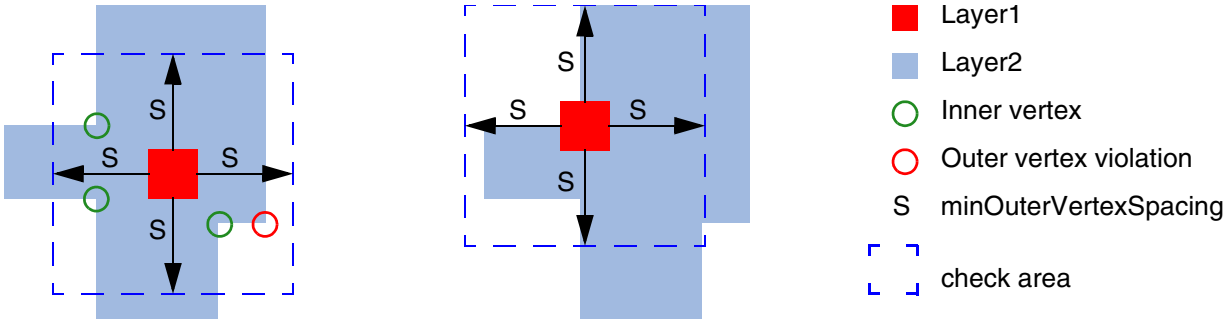
Specifies the minimum spacing between a shape on layer1 and all outer vertices of an enclosing layer2 shape, provided that the layer2 shape fully encloses the layer1 shape. This constraint is not symmetric.

minOuterVertexSpacing Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minOuterVertexSpacing` constraints have a `Value` that represents the minimum spacing, in user units.



a) Violation because there is a layer2 outer vertex inside the check area.

b) Constraint does not apply because the layer1 shape is not fully enclosed by the layer2 shape.

Examples

The following example sets the minimum spacing to 0.7 user units between the edges of `Via1` shapes and any outer vertices of enclosing `Metal1` shapes when the `Via1` shapes are fully enclosed by `Metal1`.

Format	Example
Tcl	`set_layerpair_constraint -constraint minOuterVertexSpacing \ -layer1 Via1 -layer2 Metal1 -Value 0.7`
Virtuoso	`orderedSpacings( (minOuterVertexSpacing "Via1" "Metal1" 0.7 ) ) ;orderedSpacings`

Related Topics

Spacing Constraints

minParallelSpanSpacing

Specifies the minimum spacing between two objects that depends on the span lengths of the objects which have a parallel run length greater than a given value.

minParallelSpanSpacing Quick Reference

Constraint Type	Layer
Value Types	TwoDTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

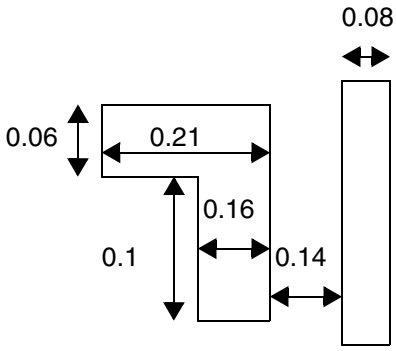
`minParallelSpanSpacing` constraints have a TwoDTblValue that represents the minimum spacing between the two objects and depends on the span lengths of the objects. The NxN two-dimensional table implicitly has the same span lengths for row and column headings. The span length values must increase in value from top to bottom in the table. The spacing values must be the same or increasing from left to right and from top to bottom in the table. Given two objects with spanLength1 and spanLength2, find the last row where spanLength1 is greater than the table row spanLength, and the right-most column where spanLength2 is greater than the table column spanLength, the intersection of the matching row and column is the required spacing.

Parameters

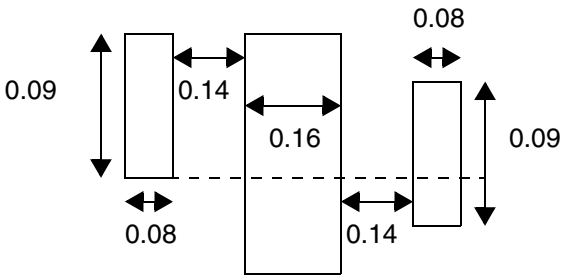
- `length` (Value) The constraint applies when the parallel run length between the two objects is greater than this value.

Examples

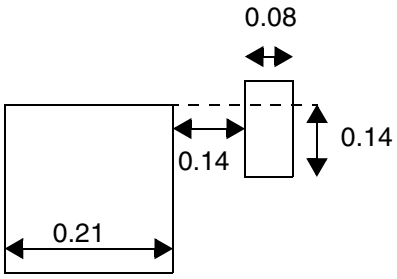
The following figures show how the constraint is interpreted in this example.



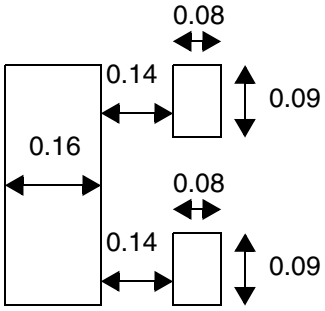
a) Violation, the continuous parallel run length of 0.1 and 0.06 = 0.16 (> 0.15), and top and bottom edges on the left have a span length > 0.1, so 0.15 spacing is required



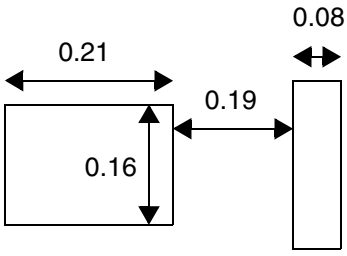
b) OK, parallel run length does not count for opposite sides



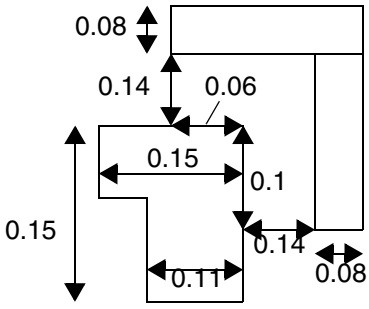
c) OK, parallel run length < 0.15



d) OK, parallel run length is not continuous, and each of the individual PRLs of 0.09 <= 0.15 would not trigger the rule



e) Violation, the parallel run length >0.15, so the constraint applies and requires a spacing of 0.20 because the left shape's span length is 0.21 (> 0.20)



f) OK, the top and side parallel run lengths are calculated separately, and each is <= 0.15

Format	Example

Tcl	`set_constraint_parameter -name length -Value 0.15`
	`set_layer_constraint -constraint minParallelSpanSpacing \`
	`  -layer Metal2 -hardness hard \`
	`  -TblCols  {0  0.10  0.20} \`
	`  -TwoDTblValue {  0 0.10  0.15  0.20  \`
	`  0.10 0.15  0.17  0.23  \`
	`  0.20 0.20  0.23  0.25 }`

LEF	`PROPERTY LEF58_SPACINGTABLE`
	`"SPACINGTABLE`
	`PARALLELSPANLENGTH PRL 0.15`
	`#SPANLENGTH=  0  0.10  0.20`
	`SPANLENGTH  0 0.10  0.15  0.20`
	`SPANLENGTH  0.10 0.15  0.17  0.23`
	`SPANLENGTH  0.20 0.20  0.23  0.25 ;" ;`

Related Topics

Spacing Constraints

minProtrudedProximitySpacing

Sets the minimum spacing between a protrusion from a wide wire and shapes that fall within a specified distance of a shape of a specified width. The protrusion contributes to the proximity halo of the larger shape. The smaller shapes effectively pick up the spacing required by the wider shape.

minProtrudedProximitySpacing and minProximitySpacing are mutually exclusive. If both are specified, then only minProtrudedProximitySpacing is honored.

minProtrudedProximitySpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`, `OneDTblValue`, `TwoDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Types

`minProtrudedProximitySpacing` constraints have the following value types:

- `Value`

  Specifies the minimum spacing, in user units, required between the protrusion and the small geometry. *This value type is not currently supported by Space-based Router and Chip Optimizer functions.*

- `OneDTblValue`

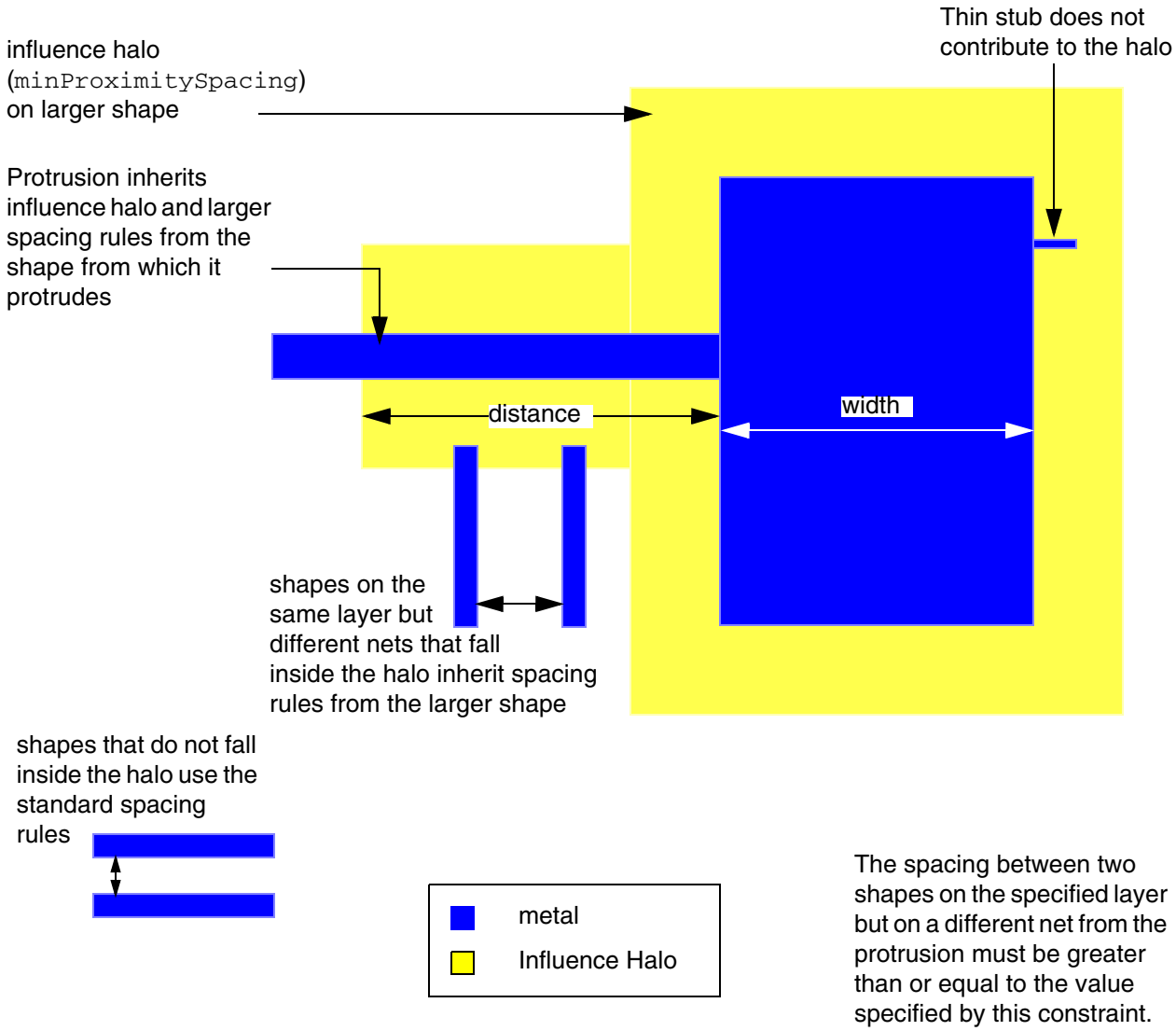
  The table values represent the minimum spacing, in user units, required between the protrusion and the small geometry, based on the row lookup key (`width`) for the width of the large influencing shape. The width of the shape is defined as the smaller of the shape's two dimensions. *This value type is not currently supported by Space-based Router and Chip Optimizer functions.*

- `TwoDTblValue`

  The table values represent the minimum spacing, in user units, required between the protrusion and the small geometry, based on the row lookup key (`width`) for the width of the large influencing shape, and the column lookup key (`distance`) that represents the distance between the small shape and the large influencing shape.

To use this constraint with VirtuosoSpace-basedRouter functions, minSpacing must also be set to a `OneDTblValue` or `TwoDTblValue`.

Format	Example
Tcl	`set_layer_constraint -constraint minProtrudedProximitySpacing \ -layer s_layer -row_name width -col_name length \ -TblCols {f_distance…} -TwoDTblValue {{f_width f_space}…}`



The spacing between two shapes on the specified layer but on a different net from the protrusion must be greater than or equal to the value specified by this constraint.

Examples

This example sets the minProtrudedProximitySpacing constraint.

Format	Example

Tcl	`set_layer_constraint -layer Metal2`
	`  -constraint minProtrudedProximitySpacing -hardness hard \`
	`  -row_name width -col_name distance \`
	`  -TblCols { 0.15 } \`
	`  -TwoDTblValue { 0.299 0.15 }  \`
	`  -row_interpolation snap_down_inclusive \`
	`  -row_extrapolation {snap_down snap_down} \`
	`  -col_interpolation snap_down_inclusive \`
	`  -col_extrapolation {snap_up snap_down}`

LEF	`SPACING 0.15 RANGE 0.299 10000 INFLUENCE 0.15 ;`

Virtuoso	`( minStubInfluenceSpacing "Metal2"`
	`  (( "width" nil nil "distance" nil nil ) )`
	`   (`
	`   (0.299  0.15 ) 0.15`
	`   )`
	`   )`

Related Topics

Spacing Constraints

minProtrusionSpacing

Specifies the required spacings between a wide wire and neighbor wires for the following conditions:

- The wide wire width is greater than a specified width (`width`).
- The neighbor wires have a parallel run length greater than a specified length (`parallelRunlength`).
- The neighbor wires are within a specified distance (`oaDistanceWithin`) of the wide wire.

The constraint applies under the following conditions:

- If there is a jog/protrusion wire/edge between two convex corners in either the wide wire or the neighbor wires with continuous common projected length less than or equal to (<=) the specified jog width (`length`), then the spacing between opposite metal shapes must be greater than or equal to the constraint's short jog spacing.
- If the continuous common projected length is greater than (>) the specified jog width (`length`), then the spacing between opposite metal shapes must be greater than or equal to the constraint's long jog spacing.
- If there are two or more protrusion wires or edges with continuous common projected length that is less than or equal to (<=) the jog width (`length`), and the spacing on the protrusion wires is less than (<) the long jog spacing, then the spacing between any two protrusion wires must be greater than or equal to the constraint's jog-to-jog spacing.

minProtrusionSpacing Quick Reference

Constraint Type	Layer
Value Types	`ValueArrayValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing
Group Operators	AND

Value Type

`minProtrusionSpacing` constraints have a ValueArrayValue of three Value elements representing, in order, *short jog spacing*, *long jog spacing*, and *jog-to-jog spacing*.

The constraint can be specified for multiple wide widths and with various `parallelRunLength`, `oaDistanceWithin`, and `length` parameters, using an AND constraint group.

To determine the spacing requirements, find the largest values for `width` and `parallelRunLength,` where the wide wire width > `width` and the neighbor wire's parallel run length > `parallelRunLength`, and the smallest value `oaDistanceWithin` where the neighbor wire is < `oaDistanceWithin` away from the wide wire. If `excludeSpacing` is given, wires that are within the given distance will be ignored.

- If there is a jog/protrusion wire/edge between two convex corners in either the wide wire or the neighbor wires with ***continuous common projected length*** less than or equal to (<=) the specified jog width (`length`), then the spacing between opposite metal shapes must be greater than or equal to the constraint's short jog spacing.
- If the continuous common projected length is greater than (>) the specified jog width (`length`), then the spacing between opposite metal shapes must be greater than or equal to the constraint's long jog spacing.
- If there are two or more protrusion wires or edges with continuous common projected length that is less than or equal to (<=) the jog width (`length`), and the spacing on the protrusion wires is less than (<) the long jog spacing, then the spacing between any two protrusion wires must be greater than or equal to the constraint's jog-to-jog spacing.

The following figure shows the syntax for setting the `minProtrusionSpacing` constraint.

Syntax for minProtrusionSpacing

Format	Syntax

Tcl	`set_constraint_parameter -name width -Value f_width`
	`set_constraint_parameter -name parallelRunLength -Value f_PRL`
	`set_constraint_parameter -name oaDistanceWithin -Value f_distance`
	`set_constraint_parameter -name length -Value f_jogWidth`
	`[set_constraint_parameter -name excludeSpacing -RangeValue s_range]`
	`set_layer_constraint -constraint minProtrusionSpacing \`
	`  -layer s_layerName -ValueArrayValue { \`
	`  f_shortJogSpacing f_longJogSpacing f_jogToJogSpacing }`

LEF	`PROPERTY LEF58_SPACINGTABLE`
	`"SPACINGTABLE JOGTOJOGSPACING f_jogToJogSpacing`
	`  JOGWIDTH f_jogWidth SHORTJOGSPACING f_shortJogSpacing`
	`  {WIDTH f_width PARALLEL f_PRL WITHIN f_distance`
	`  [EXCEPTWITHIN f_lowExcludeSpacing f_highExcludeSpacing]`
	`  LONGJOGSPACING f_longJogSpacing`
	`  [SHORTJOGSPACING f_widthShortJogSpacing]}…`
	`;" ;`

Format	Syntax

Virtuoso	`spacings(`
	`  ( minProtrusionSpacing` *`s_layerName`*
	`  'width` *`f_width`*
	`  'length` *`f_PRL`*
	`  'within` *`f_distance`*
	`  'protrusionLength` *`f_jogWidth`*
	`  ['excludeSpacing (`*`s_range`*`)]`
	`   (`
	`   short` *`f_shortJogSpacing`*
	`   long` *`f_longJogSpacing`*
	`   between` *`f_jogToJogSpacing`*
	`   )`
	`  }`
	`) ;spacings`

Parameters

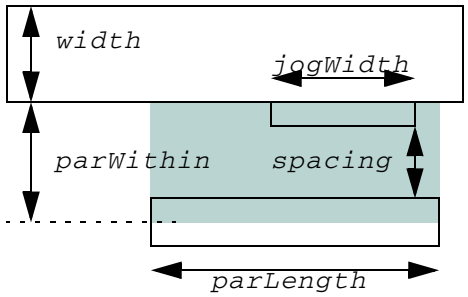
- `width` (`Value`, required) The constraint applies only when the wide wire width, in user units, is greater than this value.
- `parallelRunLength` (`Value`, required) The constraint applies only when the parallel run length, in user units, between the wide wire and the neighbor wires is greater than this value (*parLength*).
- `oaDistanceWithin` (`Value`, required) The constraint applies only when the neighbor wires are less than this distance, in user units, from the wide wire (*parWithin*).
- `length` (`Value`, required) The continuous common projected length (*jogWidth*) of the protrusion wire(s) is compared with this value to determine the constraint's value.
- `excludeSpacing` (`RangeValue`, optional) specifies that any wire within the exclusion area given by this range parameter will be ignored. No spacing violation will be reported for such wires. For example,

  ```
  set_constraint_parameter -name excludeSpacing -RangeValue {"[0.2 0.9)"}
  ```

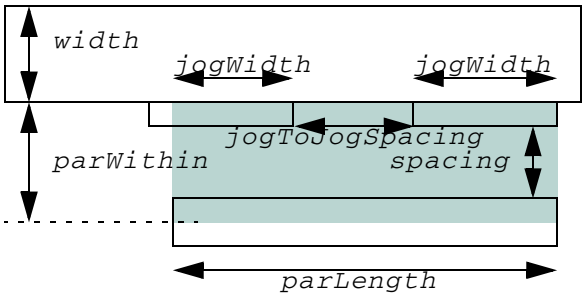
  specifies that wires that are greater than or equal to 0.2 user units and less than 0.9 user units from the wide wire will be ignored. See Example 3—minProtrusionSpacing with excludeSpacing for a graphic example.

The following figure shows how these parameters are measured for this constraint. The green shaded region shows the area being checked.

Illustration for minProtrusionSpacing Constraint



When all other conditions are met, if the protrusion length (*jogWidth*) is > `length`, then these shapes must be > long jog spacing apart. If it is <= `length`, then these shapes must be > short jog spacing apart.



When all other conditions are met and the protrusion lengths (*jogWidth*) <= `length`, and the protrusion spacings are < long jog spacing from the opposite wire, then the protrusions must be >= jog-to-jog spacing apart.

Examples

This section includes the following examples:

- Example 1—minProtrusionSpacing (Two Widths)
- Example 2—minProtrusionSpacing (One Width)
- Example 3—minProtrusionSpacing with excludeSpacing

Example 1—minProtrusionSpacing (Two Widths)

Graphic examples for `minProtrusionSpacing` commands in the following table, which set two wide widths, are shown in Illustration for minProtrusionSpacing (Two Widths).

Example 1—minProtrusionSpacing (Two Widths)

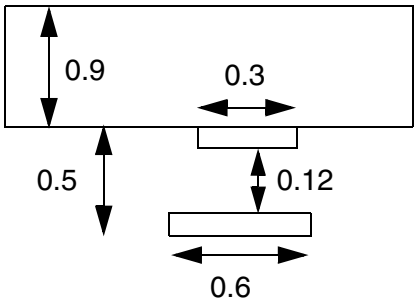
Format	Syntax

Tcl	`create_constraint_group -name proSpaceGp -opType and`
	
	`set_constraint_parameter -name width -Value 0.4`
	`set_constraint_parameter -name parallelRunLength -Value 0.2`
	`set_constraint_parameter -name oaDistanceWithin -Value 0.5`
	`set_constraint_parameter -name length -Value 0.2`
	`set_layer_constraint -constraint minProtrusionSpacing \`
	`  -layer Metal1 -group proSpaceGp -create true \`
	`  -ValueArrayValue { 0.08 0.13 0.7 }`
	
	`set_constraint_parameter -name width -Value 0.8`
	`set_constraint_parameter -name parallelRunLength -Value 0.3`
	`set_constraint_parameter -name oaDistanceWithin -Value 0.5`
	`set_constraint_parameter -name length -Value 0.2`
	`set_layer_constraint -constraint minProtrusionSpacing \`
	`  -layer Metal1 -group proSpaceGp -create true \`
	`  -ValueArrayValue { 0.08 0.15 0.7 }`

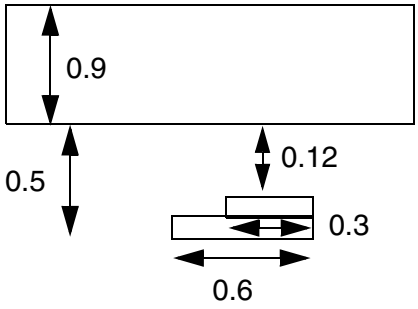
LEF	`PROPERTY LEF58_SPACINGTABLE "`
	`  "SPACINGTABLE JOGTOJOGSPACING 0.7`
	`  JOGWIDTH 0.2 SHORTJOGSPACING 0.08`
	`  WIDTH 0.4 PARALLEL 0.2 WITHIN 0.5`
	`  LONGJOGSPACING 0.13`
	`  WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5`
	`  LONGJOGSPACING 0.15 ; " ;`

Virtuoso	`spacings(`
	` ( minProtrusionSpacing "Metal1" 'width 0.4 'length 0.2`
	`  'within 0.29 'protrusionLength 0.2`
	`  (`
	`  short 0.08`
	`  long 0.13`
	`  between 0.7`
	`  )`
	` )`
	` ( minProtrusionSpacing "Metal1" 'width 0.8 'length 0.3`
	`  'within 0.5 'protrusionLength 0.2`
	`  (`
	`  short 0.08`
	`  long 0.15`
	`  between 0.7`
	`  )`
	` )`
	`) ;spacings`

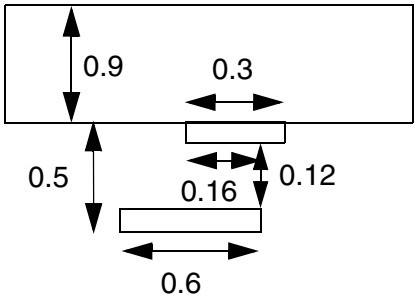
Illustration for minProtrusionSpacing (Two Widths)



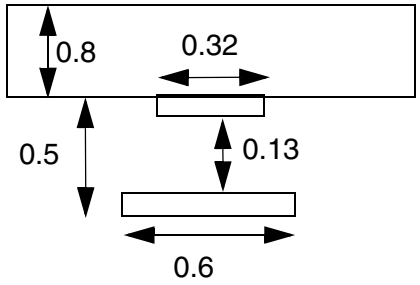
a) Violation, all the triggering conditions are met, and the jog spacing is 0.12 (< 0.15 required long jog spacing).



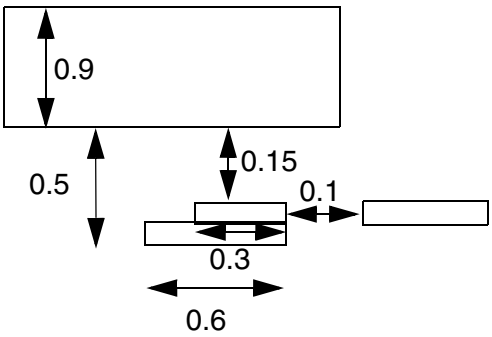
b) Violation, same conditions as a) but the triggering protrusion is on the neighbor wire.



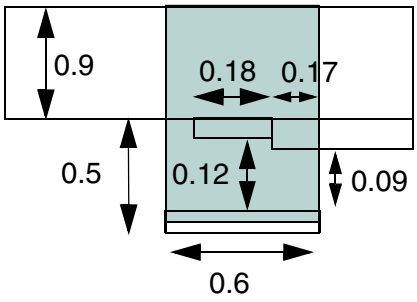
c) OK, the length of the jog/protrusion is 0.16 (< 0.2), and the spacing of 0.12 >= 0.08 (required short jog spacing).



d) OK, the wide wire without the jog/ protrusion must have width > 0.8 to trigger the spacing of 0.15. Here, the spacing of 0.13 >= 0.13 (required long jog spacing based on width > 0.4).



e) OK, the 0.15 spacing is applied only to the opposite metal, but not to the right side neighbor of 0.10 away.



f) Violation, the continuous common projected length with spacing < 0.15 is 0.35 (0.18 + 0.17), which is > 0.2 and requires a spacing of 0.15.

Example 2—minProtrusionSpacing (One Width)

Graphic examples for minProtrusionSpacing commands in the following table, which set one width, are shown in figures Illustration for minProtrusionSpacing (One Width) and Additional Illustrations for minProtrusionSpacing (One Width).

Table 16-1 Example 2—minProtrusionSpacing (One Width)

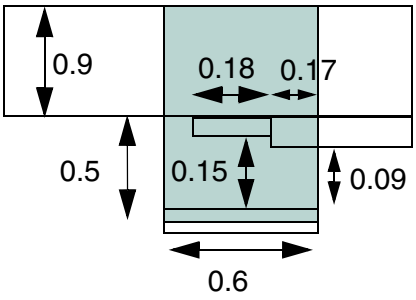
Format	Syntax

Tcl	`set_constraint_parameter -name width -Value 0.8`
	`set_constraint_parameter -name parallelRunLength -Value 0.3`
	`set_constraint_parameter -name oaDistanceWithin -Value 0.5`
	`set_constraint_parameter -name length -Value 0.2`
	`set_layer_constraint -constraint minProtrusionSpacing`
	`  -layer Metal1 -ValueArrayValue { 0.08 0.15 0.7 }`

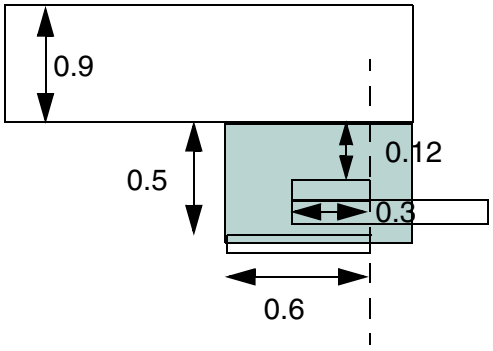
LEF	`PROPERTY LEF58_SPACINGTABLE`
	`"SPACINGTABLE JOGTOJOGSPACING 0.7 JOGWIDTH 0.2 SHORTJOGSPACING 0.08`
	`  WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5 LONGJOGSPACING 0.15 ;" ;`

Virtuoso	`spacings(`
	` ( minProtrusionSpacing "Metal1" 'width 0.8 'length 0.3`
	`    'within 0.5 'protrusionLength 0.2`
	`   (`
	`   short 0.08`
	`   long 0.15`
	`   between 0.7`
	`   )`
	` )`
	`) ;spacings`

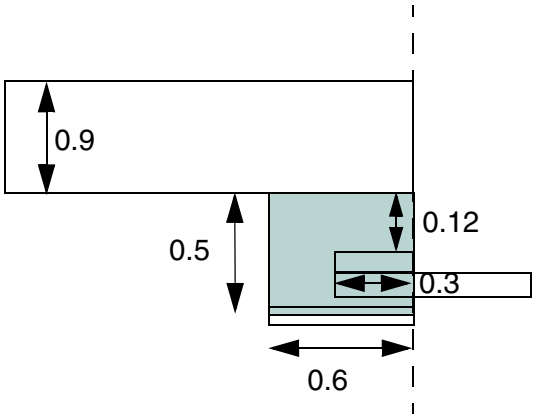
Illustration for minProtrusionSpacing (One Width)



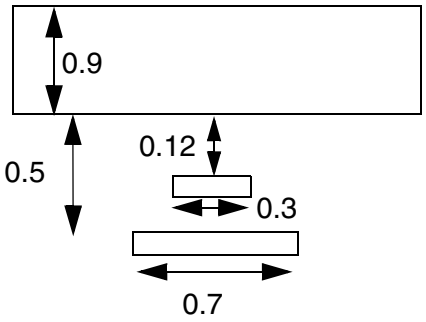
a) OK, only 0.17 segment is within 0.15, spacing of 0.08 is required and met.



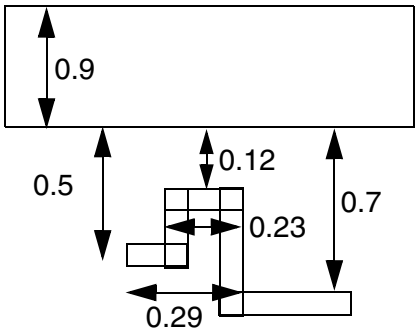
b) Violation, all neighbor wires in the green region are considered. A long jog spacing of 0.15 is required.



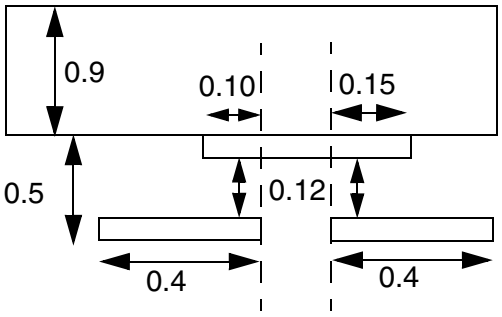
c) Violation, any neighbor wires with run length > 0.2 must have spacing >= 0.15.



d) Violation, the neighbor wire does not need to be a jog.

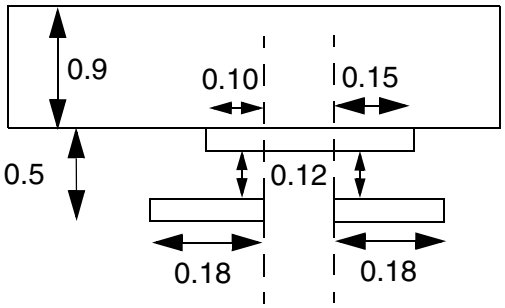


e) OK, the common projected length is 0.29 within the 0.5 away, the rule is not checked.

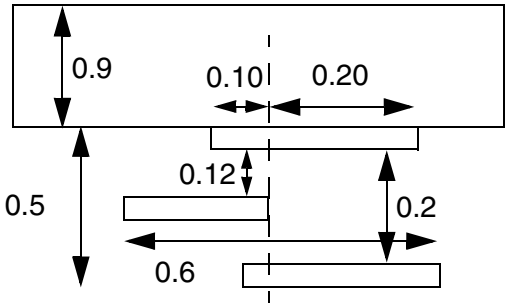


f) OK, the common projected lengths of 0.10 and 0.15 individually only require 0.08 spacing and are met.

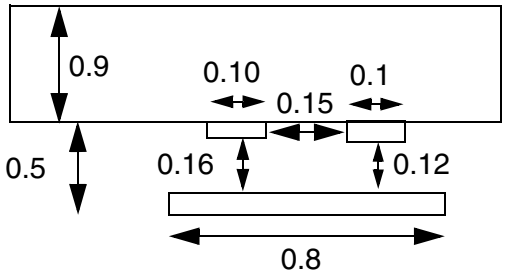
Additional Illustrations for minProtrusionSpacing (One Width)



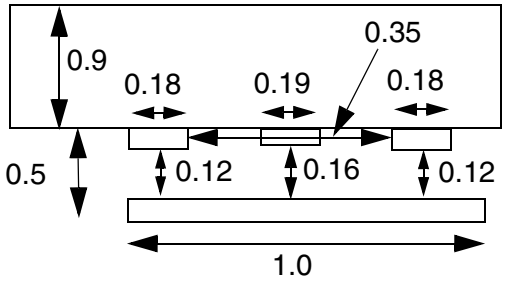
a) OK, parallel run length of 0.18 individually would not trigger the rule.



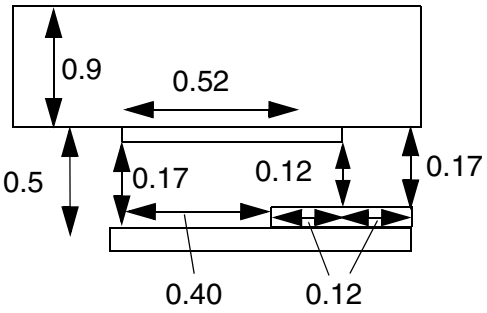
b) OK, only the 0.10 segment is within 0.15, spacing of 0.06 is required and met.



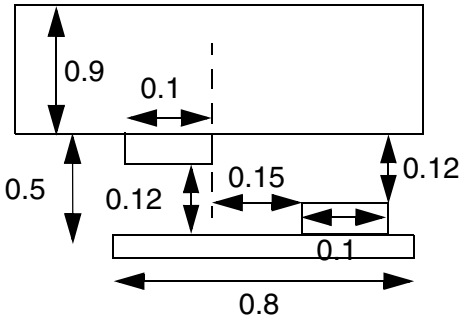
c) OK, the left jog/protrusion having spacing of 0.16 (>= 0.15) is not a valid jog/protrusion for considering spacing among jog/protrusion wires.



d) Violation, having a 'good' 0.19 jog/protrusion at the middle is irrelevant, the distance of 0.35 between the left and right jog/protrusion wires still violate >= 0.7 required jog to jog spacing.



e) OK, the top 0.52 jog/protrusion is broken into two segments for the rule, and 0.40 segment has spacing of 0.17 (>= 0.15), and the 0.12 segment has spacing of 0.12 (>= 0.08). The bottom 0.24 jog/protrusion is broken into two segments of 0.12 with spacing of 0.17 and 0.12 (>= 0.08).



f) Violation, jog/protrusion wires on opposite wires still need to follow the 0.7 spacing.

Example 3—minProtrusionSpacing with excludeSpacing

In this example, the required spacing between a wire that is within 0.5 user units of a > 0.8 wide wire with a protrusion and a parallel run length > 0.3 is dependent on the length of the protrusion (*jogWidth*). Short jog spacing is 0.08, long jog spacing is 0.15, and jog-to-jog spacing is 0.7. However, when the narrow wire is in the exclusion area, >= 0.25 and < 0.4 user units from the wide wire, the constraint does not apply.

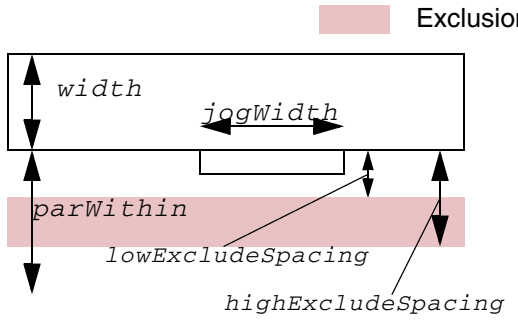
Format	Syntax

Tcl	`set_constraint_parameter -name width -Value 0.8`
	`set_constraint_parameter -name parallelRunLength -Value 0.3`
	`set_constraint_parameter -name oaDistanceWithin -Value 0.5`
	`set_constraint_parameter -name length -Value 0.2`
	`set_constraint_parameter -name excludeSpacing \`
	`  -RangeValue {"[0.25 0.4)"}`
	`set_layer_constraint -constraint minProtrusionSpacing \`
	`  -layer Metal1 -ValueArrayValue { 0.08 0.15 0.07}`

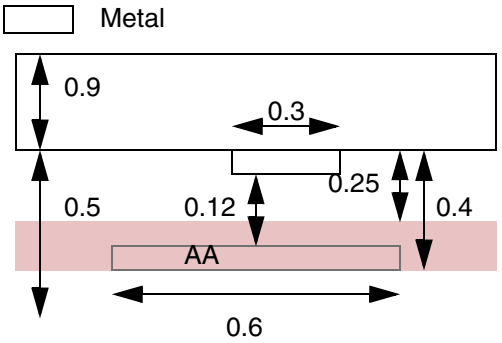
LEF	`PROPERTY LEF58_SPACINGTABLE`
	`"SPACINGTABLE JOGTOJOGSPACING 0.07 JOGWIDTH 0.2 SHORTJOGSPACING 0.08`
	`  WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5`
	`  EXCEPTWITHIN 0.25 0.4 LONGJOGSPACING 0.15 ;" ;`

Virtuoso	`spacings(`
	` ( minProtrusionSpacing "Metal1"`
	`  'width  0.8`
	`  'length 0.3`
	`  'within 0.5`
	`  'protrusionLength 0.2`
	`  'excludeSpacing {"[0.25 0.4)"}`
	`   (`
	`   short   0.08`
	`   long    0.15`
	`   between 0.07`
	`   )`
	` )`
	`) ;spacings`

Illustration for minProtrusionSpacing with excludeSpacing



Any wire within the exclusion area will be ignored. No spacing violation will be reported for such wires.



a) OK. The narrow wire AA is in the exclusion area >= 0.25 and < 0.4 user units from the wide wire, so the constraint does not apply, and the 0.12 protrusion spacing does not violate the required 0.15 long jog spacing.

Related Topics

Spacing Constraints

check_space

minProximitySpacing

Sets the minimum spacing between shapes on the specified layer that fall within a specified distance of a shape of a specified width.

Also known as proximity or influence rules, the spacing required by smaller shapes is influenced by large nearby shapes. The smaller shapes effectively pick up the spacing required by the wider neighboring shape.

minProtrudedProximitySpacing and minProximitySpacing are mutually exclusive. If both are specified, then only minProtrudedProximitySpacing is honored.

minProximitySpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`, `OneDTblValue`, `TwoDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Types

- `Value`

  Sets the minimum spacing between shapes on the specified layer. *This value type is not currently supported by Space-based Router and Chip Optimizer functions.*

- `OneDTblValue`

  The minimum spacing is the table value that changes according to the lookup key (`width`) which is the width of the wide shape. *This value type is not currently supported by Space-based Router and Chip Optimizer functions.*

- `TwoDTblValue`

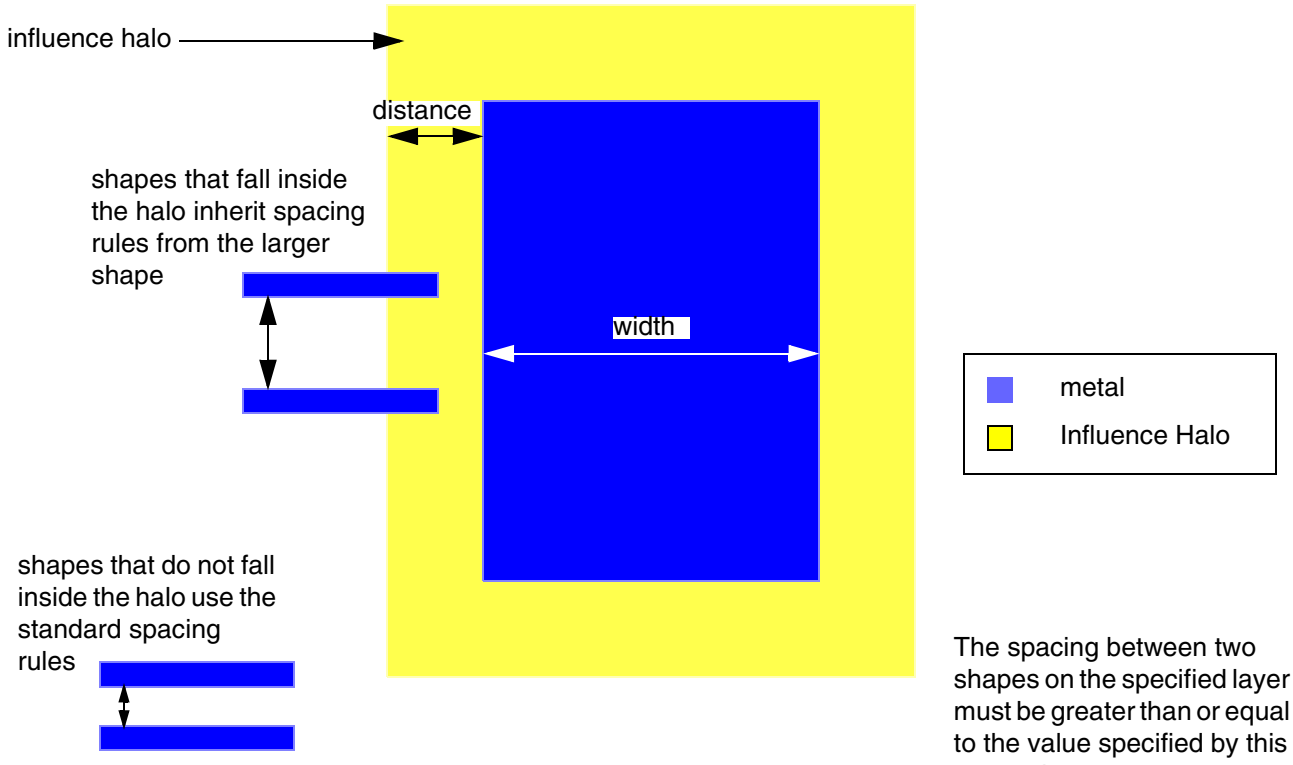
The minimum spacing is the table value that changes according to the row lookup key (`width`) which is the width of the wide shape, and the column lookup key (`distance`), which is the distance within which the small shape must be from the wide shape.

Format	Example

Tcl	`set_layer_constraint -constraint minProximitySpacing \`
	` -layer layer -row_name width -col_name distance \`
	` -TblCols {f_influenceLength …}`
	` -TwoDTblValue {{f_width {f_space…}}…}`

LEF	`LAYER layer`
	` TYPE ROUTING ;`
	` SPACING f_space`
	`  RANGE minWidth maxWidth`
	`  INFLUENCE influenceLength ;`

Virtuoso	`spacingTables(`
	`  ( minInfluenceSpacing tx_layer`
	`  ( ( "width" nil nil [ "distance" nil nil ] ) )`
	`  ( l_table )`
	`  )`
	`  )`



The spacing between two shapes on the specified layer must be greater than or equal to the value specified by this constraint.

Related Topics

Spacing Constraints

minSameNetSpacing

Sets the minimum spacing, edge-to-edge, between shapes on the specified layer and on the same net.

minSameNetSpacing Quick Reference

Constraint Type	Layer
Value Types	Value, OneDTblValue, TwoDTblValue (custom, derived)
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Types

`minSameNetSpacing` constraints have the following value types:

- Value

  Specifies the minimum spacing, edge-to-edge, between shapes on the specified layer and on the same net.

Format	Example
Tcl	`set_layer_constraint -constraint minSameNetSpacing \` `-layer` *`s_layer`* `-Value` *`f_space`*

- OneDTblValue

  Specifies the minimum spacing based on width.

Format	Example

Tcl	`set_layer_constraint -constraint minSameNetSpacing \`
	`-layer` *`s_layer`* `-row_name width \`
	`-OneDTblValue` **{**{*`f_width f_space`*}…**}**

- TwoDTblValue (custom, derived)

  See minSpacing for examples.

Parameters

- `widthLengthTableType` (`IntValue`) specifies the table type when `minSameNetSpacing` is a `TwoDTblValue` table. The values are given in Table 16-1 on page 811.
- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1). For information on this parameter, see Euclidean and Manhattan Spacing Constraints.
- `eolAddSpace` (`OneDTblValue`, optional) specifies additional spacing required based on the wire width.
- `eolRadialCheck` (`BoolValue`, optional) enables radial checking from the wire corners.
- `oaSpacingDirection` (`StringAsIntValue`, optional) specifies the measurement direction.

anyDirection	0	Horizontally and vertically (default)
horizontalDirection	1	Horizontally only
verticalDirection	2	Vertically only

Examples

- Fixed Value

  This example sets the minimum same net spacing on `poly1` to `1.0`.

Format	Example
Tcl	`set_layer_constraint -constraint minSameNetSpacing \` `-layer poly1 -Value 1.0`

Virtuoso	`spacings(`
	`( minSameNetSpacing poly1 1.0 )`
	`)`

Related Topics

Spacing Constraints

minSideSpacing

Specifies the spacing between the shapes on the same layer for a given (short or long) edge.

minSideSpacing Quick Reference

Constraint Type	Layer (non-symmetric)
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

Value	Specifies the minimum spacing between the edges of the two shapes.

Required Parameters

spacingType	Specifies the type of edges of both the shapes to be considered for spacing check. Type: IntValue 0 anySideToAnySide 1 shortSideToShortSide 2 shortSideToLongSide 3 shortSideToAnySide 4 longSideToLongSide 5 longSideToAnySide

Optional Parameters

parallelRunLength	Specifies that the constraint applies only if the parallel run length between the edges of the shapes is equal to or greater than this value. Type: Value **Note:** This constraint supports negative parallel run length.

`widthRangeArray` Specifies that the constraint applies if the width of the first shape is in this range.

Type: RangeArrayValue

`otherWidthRangeArray`

Specifies that the constraint applies if width of the second shape is in this range.

Type: RangeArrayValue

`oaSpacingDirection` Specifies the measurement direction.

Type: StringAsIntValue
0 anyDirection (horizontally and vertically, default)
1 horizontalDirection (horizontally only)
2 verticalDirection (vertically only)

Example—minSideSpacing Using Value

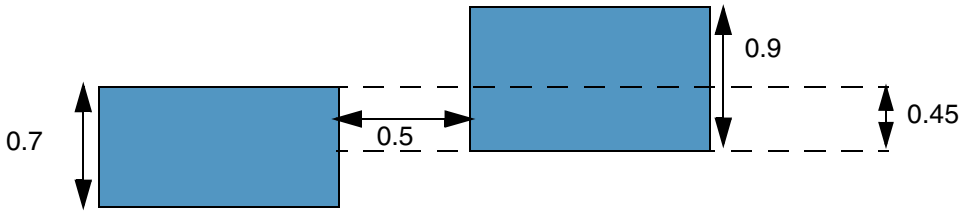
```
set_constraint_parameter -name spacingType -IntValue 1
set_constraint_parameter -name parallelRunLength -Value 0.0
set_constraint_parameter -name widthRangeArray \
  -RangeArrayValue { "[0.6 0.8]" }
set_constraint_parameter -name otherWidthRangeArray \
  -RangeArrayValue { "[0.9 1.0]" }
set_layer_constraint -constraint minSideSpacing -layer Metal1 -Value 0.4
```

Specifies that the minimum spacing between the two shapes on `Metal1` must be greater than or equal to 0.4 user units under the following conditions:

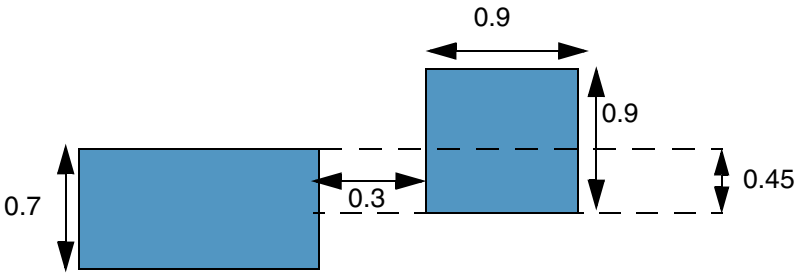
- The spacing is measured from a short side of the first shape to a short side of the second shape.
- The parallel run length between the two shapes is greater than or equal to 0.0 user units.
- The width of the first shape is greater than or equal to 0.6 and less than or equal to 0.8 user units.
- The width of the second shape is greater than or equal to 0.9 user units and less than 1.0 user units.

Illustration for minSideSpacing Using Value

```
minSideSpacing          = 0.4
  spacingType           = 1 (shortSideToShortSide)
  parallelrunLength     = 0.0
  widthRangeArray       = "[0.6 0.8]"
  otherWidthRangeArray  = "[0.9 1.0]"
```



a) PASS. The constraint applies and is met because the widths of both shapes lie in the required ranges and the spacing between shapes is 0.5 (>0.4) with non-zero parallel overlap between edges.



b) Constraint does not apply because none of the sides of a square shape is considered to be a short side.

Related Topics

Spacing Constraints

minSideSpacing

check_space

minSpacing

Determines the minimum orthogonal spacing, edge-to-edge, required between two geometries on the same layer. Spacing can be specified as the minimum spacing in user units between any two adjacent geometries, as dependent on the widths of the two geometries and, optionally, the parallel run length between the two geometries. Spacing can apply to a specific direction (horizontal or vertical), to power and ground nets only, and can use Euclidean or Manhattan measurements.

Optional parameters determine whether the constraint applies dependent on the type of connectivity or whether the shapes belong to power or ground nets.

In some processes, the spacing between the sides of shapes, when those shapes are drawn in the non-preferred direction, must be greater than the regular spacing. Spacing between an end-of-line and another end-of-line or side-of-line would be determined by an end-of-line spacing rule rather than this rule.

minSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`, `OneDTblValue`, `TwoDTblValue`
Database Types	Design, Technology
Scope	net, route, Term, instTerm, blockage, net group, group2group, reflexive, transreflexive, interChild, design, foundry
Category	Spacing
Group Operators	AND, OR

Value Types

`minSpacing` constraints have the following value types:

- `Value`

  Specifies the minimum spacing, edge-to-edge, between shapes on the specified layer.

- `OneDTblValue`

  Specifies the minimum spacing that changes according to the width of the wider of the two shapes. The width of the wider shape is the lookup key for the table. The table value is the minimum spacing. The width of a shape is defined as the smaller of the shape's two dimensions.

- `TwoDTblValue`

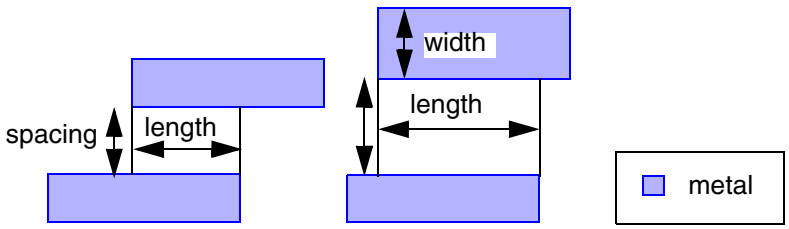
  Specifies the minimum spacing as a table of values based on variables given by the `widthLengthTableType` parameter setting. For a detailed description of these table types, see the widthLengthTableType Parameter Values table.

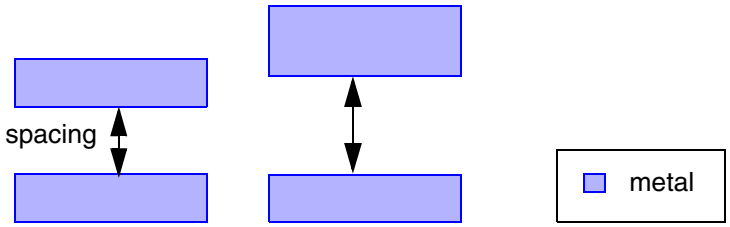
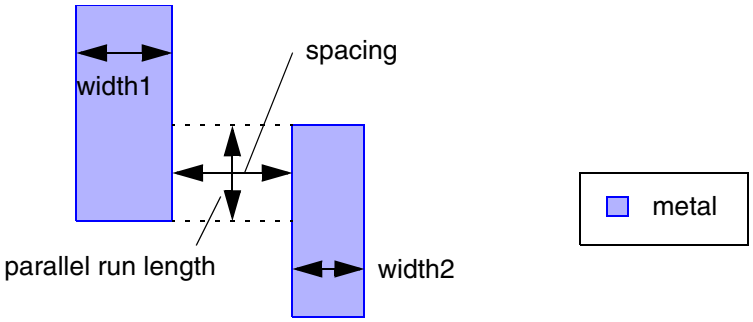
Parameters

- `widthLengthTableType` (`IntValue`) specifies the table type, as given in the widthLengthTableType Parameter Values table, when `minSpacing` is a `TwoDTblValue` table.
- `oaSpacingDirection` (`StringAsIntValue`, optional) specifies the measurement direction.

anyDirection	0	Horizontally and vertically (default)
horizontalDirection	1	Horizontally only
verticalDirection	2	Vertically only

widthLengthTableType Parameter Values

Value	row_name	col_name	Description
0	`width`	`length`	Width/Parallel run length table (default) The row lookup key represents the *width* of the wider of the two shapes, and the column lookup key represents the parallel run *length* between the two shapes. The table value represents the minimum required spacing. 

Value	row_name	col_name	Description
1	`width`	`width`	Width/width table Both the rows and columns represent the widths of the two shapes being analyzed. The table value represents the minimum required spacing. 
2	`twoWidths`	`length`	Two Width/Parallel run length table The row lookup key represents the *width* of the shapes and the column lookup key represents the parallel run *length* between the two shapes. The table value represents the minimum required spacing. 

- `oaSpacingDirection` (`StringAsIntValue`, optional) specifies the measurement direction as given in the following table.

oaSpacingDirection Values

String	Integer Value	Constraint applies:
`anyDirection`	0	Horizontally and vertically (default)
`horizontalDirection`	1	In the horizontal direction only

String	Integer Value	Constraint applies:
`verticalDirection`	2	In the vertical direction only

- `endOfLineWidth` (`Value`, optional) When set, the constraint specifies the minimum spacing between the long (or side) edges of two non-end-of-line shapes on the same layer which are aligned in the non-preferred routing direction and are greater than or equal to this parameter value in length, with a non-zero parallel run length. `oaSpacingDirection` is required when this parameter is set.
- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1). For information on this parameter, see Euclidean and Manhattan Spacing Constraints.
- `oaPGNet` (`BoolValue`, optional) If set to `true`, this constraint only applies to power and ground nets. By default and when set to `false`, this constraint applies to all nets.
- `oaConnectivityType` (`IntValue`, optional) specifies that the constraint applies only to shapes with the given connectivity, as specified in the following table.

oaConnectivityType Parameter Values for minSpacing

String	Integer Value	Constraint applies to:
`anyConnectivity`	0	(Default) Any connectivity
`sameNetConnectivity`	1	Shapes on the same net
`sameIslandConnectivity`	2	Contiguous same metal shapes

- `oaArea` (`AreaValue`, optional) specifies that the constraint only applies between any shape with area less than this value and all other shapes on the layer. If this parameter is not specified, the spacing is not dependent on shape area. No other constraint parameters may be used in combination with this constraint parameter.
- `sameMask` (`BoolValue`, optional) specifies whether the constraint applies between shapes on the same mask (`true`) or between all shapes on the given layer (`false`/ default). When `true`, the constraint specifies side-to-side or corner-to-corner spacing for advanced nodes.
- `ignoreShapesInRanges` (`RangeArray1DTblValue`, optional) parameter optionally describes a set of ranges, based on width, where the existence of shapes is allowed.

This range should lie within the minimum spacing defined for the given width in the minSpacing's table of spacing values. Shapes at spacings within these ranges are not considered violations. Shapes at spacings outside of these ranges but at less than minSpacing from a given shape violate the minSpacing constraint even if there are shapes in one of the ignoreShapesInRanges between the original shape and another shape within minSpacing. In other words, shapes with a spacing in the ignoreShapesInRanges ranges should be ignored by a design rule checker as if they did not exist.

Examples

- Example 1—minSpacing Fixed Value
- Example 2—minSpacing with 1D Table: Index width
- Example 3—minSpacing with 2D Table: Indices width and width
- Example 4—minSpacing with 2D Table: Indices width and length
- Example 5—minSpacing with 2D Table: Indices twoWidths and length
- Example 6—minSpacing with oaArea
- Example 7—minSpacing with endOfLineWidth
- Example 8 —minSpacing with ignoreShapesInRanges

Example 1—minSpacing Fixed Value

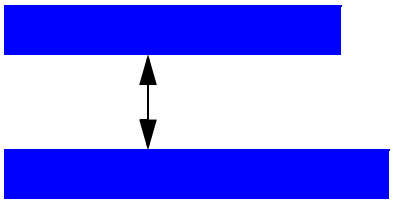
In this example, the spacing between shapes must be greater than or equal to 0.10 user units on `Metal2` as shown in the following figure.

Format	Example
Tcl	`set_layer_constraint -layer Metal2 -constraint minSpacing \` `  -hardness hard -Value .10`

LEF	`LAYER Metal2`
	`  TYPE ROUTING ;`
	`  SPACING 0.1 ;`

Virtuoso	`spacings(`
	`  (minSpacing "Metal2" 0.1)`
	`  )`

Illustration for minSpacing Fixed Value



The edge-to-edge spacing between two geometries must be greater than or equal to the `minSpacing` constraint value.

Example 2—minSpacing with 1D Table: Index width

This example establishes a lookup table that is used to set the minimum spacing for a shape on `Metal2`, based on the width of the shape as shown in the following figure.

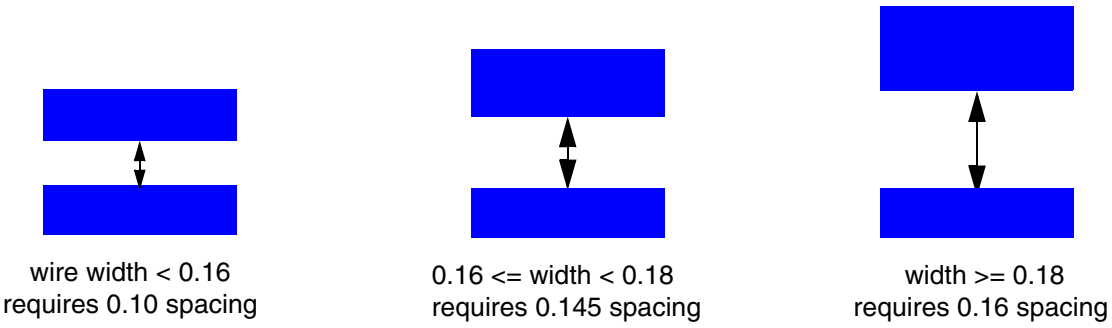
Format	Example

Tcl	`set_layer_constraint -layer Metal2 -constraint minSpacing \`
	`  -hardness hard -row_name width \`
	`  -OneDTblValue {0.000 0.100 \`
	`  0.160 0.145 \`
	`  0.180 0.160 } \`
	`  -row_interpolation snap_down \`
	`  -row_extrapolation {snap_up snap_down} \`
	`  -col_interpolation snap_down \`
	`  -col_extrapolation {snap_up snap_down}`

LEF	`SPACING 0.1 ;`
	`SPACING 0.145 RANGE 0.16 0.179 ;`
	`SPACING 0.16  RANGE 0.18 10000 ;`

Virtuoso	`( minSpacing "Metal2"`
	`  (( "width"  nil  nil )  )`
	`  (`
	`  0.0  0.1`
	`  0.16  0.145`
	`  0.18  0.16`
	`  )`
	`)`

Illustration for minSpacing OneDTblValue



Example 3—minSpacing with 2D Table: Indices width and width

This example sets the minimum spacing on `Metal2` according to the widths of two shapes on `Metal2`.

- The minimum spacing is `0.100` µm.
- If the width of one shape is greater than or equal to `0.16`, then the minimum spacing is `0.140` µm.
- If the width of both shapes is greater than or equal to `0.16`, then the minimum spacing is `0.145` µm.
- If the width of one shape is greater than or equal to `0.18`, then the minimum spacing is `0.160` µm.
- If the width of one shape is greater than or equal to `0.18`, and the width of the other shape is greater than or equal to 0.16, then the minimum spacing is `0.165` µm.
- If the width of both shapes is greater than or equal to `0.18`, then the minimum spacing is `0.180` µm.

Format	Example

Tcl	`set_constraint_parameter -name widthLengthTableType -IntValue 1`
	`set_layer_constraint -layer Metal2 -constraint minSpacing \`
	`  -hardness hard -row_name width -col_name width \`
	`  -TblCols {  0.000 0.160 0.180 } \`
	`  -TwoDTblValue {0.000 0.100 0.140 0.160 \`
	`  0.160 0.140 0.145 0.165 \`
	`  0.180 0.160 0.165 0.180 } \`
	`  -row_interpolation snap_down \`
	`  -row_extrapolation {snap_up snap_down} \`
	`  -col_interpolation snap_down \`
	`  -col_extrapolation {snap_up snap_down}`

Format	Example

LEF	`SPACINGTABLE TWOWIDTHS`
	`# width =  0.000 0.159 0.179`
	`  WIDTH 0.000 0.100 0.140 0.160`
	`  WIDTH 0.159 0.140 0.145 0.165`
	`  WIDTH 0.179 0.160 0.165 0.18 ;`

Virtuoso	`( minSpacing "Metal2"`
	`  (( "width"  nil  nil  "width"  nil  nil  )  )`
	`   (`
	`    (0.0  0.0  ) 0.100`
	`    (0.0  0.16 ) 0.140`
	`    (0.0  0.18 ) 0.160`
	`    (0.16 0.0  ) 0.140`
	`    (0.16 0.16 ) 0.145`
	`    (0.16 0.18 ) 0.165`
	`    (0.18 0.0  ) 0.160`
	`    (0.18 0.16 ) 0.165`
	`    (0.18 0.18 ) 0.180`
	`   )`
	`)`

The Virtuoso technology file ASCII data is interpreted the same as the Space-based Router and Chip Optimizer set_layer_constraint command with `-row_interpolation` and `-col_interpolation` set to **`snap_down`**. LEF width numbers are slightly different because the LEF data is interpreted the same as Space-based Router and Chip Optimizer set_layer_constraint command with `-row_interpolation` and `-col_interpolation` set to **`snap_down_inclusive`**. For more information, see Interpolation and Extrapolation Techniques.

Example 4—minSpacing with 2D Table: Indices width and length

This example sets the minimum spacing on `Metal2` according to the maximum width of two shapes and the parallel run length between them.

Minimum spacing	0.15 μm
Either width > 0.25 μm and parallel length > 0.5 μm	0.20 μm spacing
Either width > 1.50 μm and parallel length > 0.5 μm	0.50 μm spacing
Either width > 3.00 μm and parallel length > 3.00 μm	1.00 μm spacing
Either width > 5.00 μm and parallel length > 5.00 μm	2.00 μm spacing

Format **Example**

Tcl

```
set_constraint_parameter -name widthLengthTableType -IntValue 0
set_layer_constraint -layer Metal2 -constraint minSpacing \
  -hardness hard -row_name width -col_name length \
  -TblCols           {0.00  0.50  3.00  5.00} \
  -TwoDTblValue {0.00 0.15  0.15  0.15  0.15 \
                 0.25 0.15  0.20  0.20  0.20 \
                 1.50 0.15  0.50  0.50  0.50 \
                 3.00 0.15  0.50  1.00  1.00 \
                 5.00 0.15  0.50  1.00  2.00 } \
  -row_interpolation snap_down_inclusive \
  -row_extrapolation {snap_up snap_down} \
  -col_interpolation snap_down_inclusive \
  -col_extrapolation {snap_up snap_down}
```

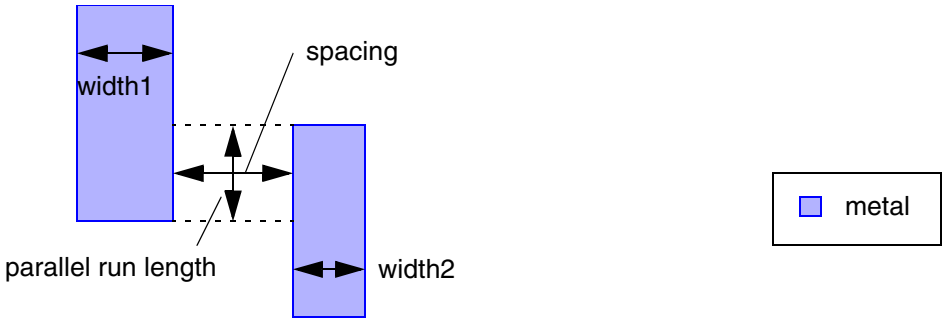
LEF

```
SPACINGTABLE
  PARALLELRUNLENGTH 0.00  0.50  3.00  5.00
  WIDTH 0.00        0.15  0.15  0.15  0.15   #max width>0.00
  WIDTH 0.25        0.15  0.20  0.20  0.20   #max width>0.25
  WIDTH 1.50        0.15  0.50  0.50  0.50   #max width>1.50
  WIDTH 3.00        0.15  0.50  1.00  1.00   #max width>3.00
  WIDTH 5.00        0.15  0.50  1.00  2.00 ; #max width>5.00
```

Virtuoso

```
( minSpacing  "Metal2"
  (( "width"   nil  nil  "length"  nil  nil  )  )
  (
    (0.0    0.0   ) 0.15
    (0.251  0.501 ) 0.20
    (1.501  0.501 ) 0.50
    (3.001  3.001 ) 1.00
    (5.001  5.001 ) 2.00
  )
)
```

Illustration for minSpacing with 2D Table: Indices width and length



The Virtuoso Space-based Router set_layer_constraint command with `-row_interpolation` and `-col_interpolation` set to **`snap_down_inclusive`** uses the same width and length numbers as LEF, whereas the Virtuoso syntax is interpreted the same as the Virtuoso Space-based Router set_layer_constraint command with `-row_interpolation` and `-col_interpolation` set to **`snap_down`**. For more information, see Interpolation and Extrapolation Techniques.

Example 5—minSpacing with 2D Table: Indices twoWidths and length

This example sets a minimum spacing constraint between two shapes which depends on the width of both shapes and the parallel run length between the shapes with the following rules:

Minimum spacing	0.15 μm
Either width >0.25 μm and parallel length > 0.0 μm	0.20 μm spacing
Both width > 0.25 μm and parallel length > 0.0 μm	0.25 μm spacing
Either width > 1.00 μm and parallel length > 1.50 μm	0.50 μm spacing
Both width > 1.00 μm and parallel length > 1.50 μm	0.60 μm spacing
Either width > 2.00 μm and parallel length > 3.00 μm	1.00 μm spacing
Both width > 2.00 μm and parallel length > 3.00 μm	1.20 μm spacing

Format **Example**

Tcl

```
set_constraint_parameter -name widthLengthTableType -IntValue 2
set_layer_constraint -layer Metal2 -constraint minSpacing \
  -hardness hard -row_name twoWidths -col_name length \
#  width=              0.00  0.25  1.00  2.00 \
  -TblCols {          -0.001 0.00  1.50  3.00} \
  -TwoDTblValue { 0     0.15  0.20  0.50  1.00 \
                  0.25  0.20  0.25  0.50  1.00 \
                  1.00  0.50  0.50  0.60  1.00 \
                  2.00  1.00  1.00  1.00  1.20 } \
  -row_interpolation snap_down_inclusive \
  -row_extrapolation {snap_up snap_down} \
  -col_interpolation snap_down_inclusive \
  -col_extrapolation {snap_up snap_down}
```

LEF

```
SPACINGTABLE TWOWIDTHS
#        width=     0.00  0.25  1.00 2.00
#          prl=     none  0.00  1.50 3.00
WIDTH 0             0.15  0.20  0.50 1.00
WIDTH 0.25 PRL 0.0  0.20  0.25  0.50 1.00
WIDTH 1.00 PRL 1.50 0.50  0.50  0.60 1.00
WIDTH 2.00 PRL 3.00 1.00  1.00  1.00 1.20 ;
```

Format	Example

Virtuoso	`spacingTables(`
	`  ( minSpacing  "Metal2"`
	`  ( ( "twoWidths"  nil nil  "length"  nil  nil  )  )`
	`  (`
	`  (0.0  none  )  0.15`
	`  (0.0  0.0  )  0.20`
	`  (0.0  1.5  )  0.50`
	`  (0.0  3.0  )  1.00`
	`  (0.25  none  )  0.20`
	`  (0.25  0.0  )  0.25`
	`  (0.25  1.5  )  0.50`
	`  (0.25  3.0  )  1.00`
	`  (1.00  none  )  0.50`
	`  (1.00  0.0  )  0.50`
	`  (1.00  1.5  )  0.60`
	`  (1.00  3.0  )  1.00`
	`  (2.00  none  )  1.00`
	`  (2.00  0.0  )  1.00`
	`  (2.00  1.5  )  1.00`
	`  (2.00  3.0  )  1.20`
	`  )`
	`  )`
	`)`

Example 6—minSpacing with oaArea

The following example sets minimum spacing for shapes with area less than 0.03 user units^2 to 0.15 user units, otherwise a minimum spacing of 0.10 user units is required.

Format	Example

Tcl	`create_constraint_group -name spGrp -opType and`
	`set_layer_constraint -layer Metal2 -constraint minSpacing \`
	`  -group spGrp -hardness hard -Value 0.10 -create true`
	`set_constraint_parameter -name oaArea -AreaValue 0.03`
	`set_layer_constraint -layer Metal2 -constraint minSpacing \`
	`  -group spGrp -hardness hard -Value 0.15 -create true`

LEF	`SPACING 0.10 ;`
	`PROPERTY LEF58_SPACING`
	`  "SPACING 0.15 AREA 0.03 ;" ;`

Virtuoso	`spacings(`
	`  (minSpacing "Metal2"`
	`  'area 0.03`
	`  0.1 )`
	`) ;spacings`

Illustration for minSpacing with oaArea

a) Violation. 0.15 spacing is required for shapes with area < 0.03 user units^2

b) OK. Both shapes have area >= 0.03 and minimum spacing of 0.10 is met.

Example 7—minSpacing with endOfLineWidth

The following wrong direction spacing rule indicates that the long/side edges between two horizontal wires must have a spacing of 0.12 µm if they are not EOL edges with length less than 0.15 µm and they have common parallel run length greater than 0. Otherwise, 0.10 µm is the minimum spacing.

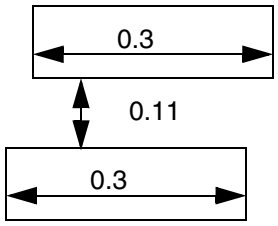
Format	Example

Tcl	`create_constraint_group -name spGrp -opType and`
	`set_layer_constraint -layer Metal2 -constraint minSpacing \`
	`  -group spGrp -hardness hard -Value 0.10 -create true`
	`set_constraint_parameter -name oaSpacingDirection \`
	`  -StringAsIntValue horizontalDirection`
	`set_constraint_parameter -name endOfLineWidth -Value 0.15`
	`set_layer_constraint -layer Metal2 -constraint minSpacing \`
	`  -group spGrp -hardness hard -Value 0.12 -create true`

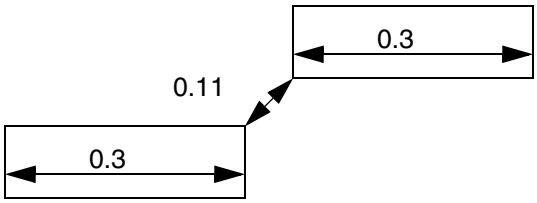
LEF	`DIRECTION VERTICAL ;`
	`SPACING 0.10 ;`
	`PROPERTY LEF58_SPACING`
	`  "SPACING 0.12 WRONGDIRECTION NONEOL 0.15 ;" ;`

Virtuoso	`spacings(`
	`  (minSpacing "Metal2"`
	`  'vertical`
	`  'eolWidth 0.15`
	`  0.12 )`
	`) ;spacings`

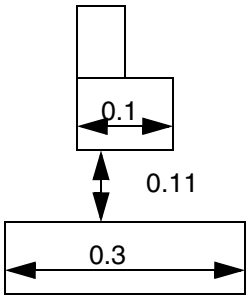
Illustration for minSpacing with endOfLineWidth



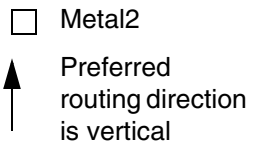
a) Violation. 0.12 spacing is required between side edges of the horizontal wires with length > 0.15 and parallel run length > 0.



b) OK. The side edges of the horizontal shapes do not have a parallel run length > 0, so minimum spacing of 0.10 is required and met.



c) OK. The horizontal edge of the top shape < 0.15 (endOfLineWidth) so the minimum spacing of 0.10 is required and met.



Example 8 —minSpacing with ignoreShapesInRanges

In this example, the shapes which are present in the range of greater than or equal to 0.07µm and less than 0.1µm for width >= 0.1µm are not considered for minimum spacing.

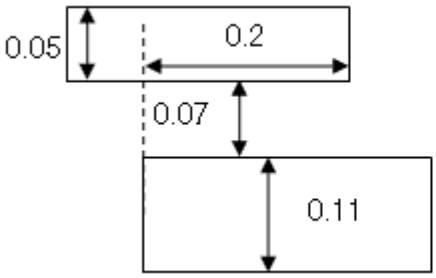
Minimum spacing	0.05µm
For PRL > -0.04 µm	0.06µm spacing
For either width > 0.1µm and PRL > 0.08µm	0.14µm spacing

Illustration for minSpacing with ignoreShapesInRanges

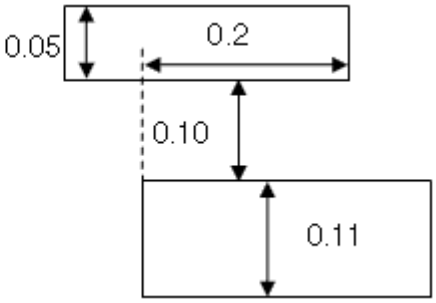
Format	Example

Tcl	`set_constraint_parameter -name ignoreShapesInRanges`
	`-RangeArray1DTblValue {0.1 1 "[0.7 0.10)"}`
	`set_layer_constraint -layer Metal2 -constraint minSpacing -hardness hard`
	`-row_name twoWidths -col_name length`
	`-TblCols {  -0.041 -0.04  0.08}`
	`-TwoDTblValue {0.0  0.05  0.06  0.06`
	`0.1  0.05  0.06  0.14}`
	`-row_interpolation snap_down_inclusive`
	`-row_extrapolation {snap_up snap_down}`
	`-col_interpolation snap_down_inclusive`
	`-col_extrapolation {snap_up snap_down}`

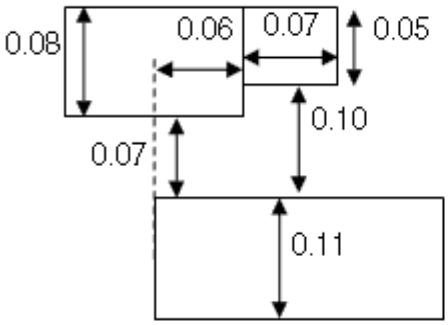
```
PRL          -0.041 -0.04  0.08
width = 0.0,   0.05  0.06  0.06
width = 0.1,   0.05  0.06  0.14
ignoreShapesInRanges = [0.07 0.10)
Width = 0.1
```



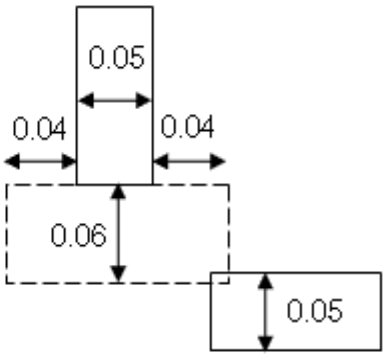
a) OK, the wire is within the exception range



b) Violation, the wire is outside the exception range



c) OK, the portion of 0.11 shape with PRL =0.06 has spacing of 0.07, which is within exception range of >= 0.07 and < 0.10. The portion of 0.11 shape with PRL=0.07 has spacing of 0.10, which is not within the exception range, so the 0.06 minSpacing applies and is met for PRL <= 0.08 and width > 0.1. A violation would have occurred if that PRL was > 0.08 which would require 0.14 minSpacing.



d) Violation, the neighbor wire is within 0.04 (PRL>-0.04), and although the corner to corner spacing would be > 0.06 (required minSpacing), the spacing is checked in MAXXY style. Any shape within the dotted area is in violation, so the constraint would fail.

Related Topics

Spacing Constraints

minSpacing (one-layer)

check_space

minVoltageSpacing

Specifies the minimum spacing between two wire shapes on the same metal layer which have different voltages.

minVoltageSpacing Quick Reference

Constraint Type	Layer
Value Types	FltHeader1DTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`minVoltageSpacing` constraints have a FltHeader1DTblValue that consists of FltValue-Value pairs where FltValue represents the voltage and Value is the minimum spacing, in user units.

To determine the minimum spacing, the maximum voltage swing between shapes is calculated as (`max`(max voltage of each shape) - (`min`(min voltage of each shape))). This voltage is compared with the voltage values in the table to find the first value that is less than or equal to the calculated voltage swing. The corresponding minimum spacing is required.

`minVoltageSpacing` constraints can be applied to layer purpose pairs and nets. For more information on these applications, see Voltage-Dependent Rule Support.

Examples

For `Metal1` shapes in this example,

- For voltage swings >= 0.0 and < 1.5, the minimum spacing is 0.06.
- For voltage swings >= 1.5 and < 3.3, the minimum spacing is 0.08.

- For voltage swings >= 3.3, the minimum spacing is 0.1.

Format	Example

Tcl	`set_layer_constraint -constraint minVoltageSpacing \`
	` -layer Metal1 -hardness hard -row_name voltage \`
	` -FltHeader1DTblValue { 0.0 0.06  1.5 0.08  3.3 0.1 } \`
	` -row_interpolation snap_down \`
	` -row_extrapolation {snap_up snap_down}`

LEF	`LAYER Metal1`
	`PROPERTY LEF58_VOLTAGESPACING`
	`  "VOLTAGESPACING`
	`   0.0 0.06`
	`   1.5 0.08`
	`   3.3 0.1 ;"`

Virtuoso	`spacingTables(`
	`  ( minVoltageSpacing "Metal1"`
	`   ( "voltage" nil nil )`
	`   ( 0.0 0.06`
	`     1.5 0.08`
	`     3.3 0.1`
	`   )`
	`  )`
	`);spacingTables`

Related Topics

Spacing Constraints

oaAllowedSpacingRange

Specifies the allowed set of spacing ranges between any two shapes on a layer. Spacing can be independent of the width of the shapes, dependent on the width of one of the shapes, or dependent on the width of both the shapes. The constraint is defined at the process level and may not apply in some cases.

The following three conditions define the bounding criteria for the constraint. At least one of these conditions must be specified to limit the bounds of the constraint:

- The constraint value range must be closed on the right-hand side, for example, "`<x`", `(x, y)`, or `[x, y]`.
- The `'overLayer` parameter must be specified, so that spacing is checked only as far as the "over layer" shape is present.
- The `'stepRange` parameter must be specified, so that the limit for discrete spacing checks is defined.

oaAllowedSpacingRange Quick Reference

Constraint Type	Layer
Value Types	RangeArrayValue, RangeArray1DTblValue, RangeArray2DTblValue
Database Types	Technology
Scope	design, foundry
Category	Spacing

Value Types

RangeArrayValue	Specifies the allowed spacing ranges for the layer.
RangeArray1DTblValue	Specifies a one-dimensional table indexed on width (parallel run length) and the allowed spacing range for that width value on the layer. The table displays records in ascending order of the index values.
RangeArray2DTblValue	Specifies a two-dimensional table that determines the value of the allowed spacing range for the given width-value pairs.

Optional Parameters

`oaSpacingDirection`	Specifies the measurement direction. `anyDirection` 0 (default) `horizontalDirection` 1 `verticalDirection` 2 Type: `StringAsIntValue`
`widthRangeArray`	Specifies that the constraint applies only to shapes whose widths are in this range. Type: `RangeArrayValue`, in user units
`parallelRunLengthTable`	Specifies that the parallel run length between the two shapes, and not the width, is the index for the one-dimensional table. The default value is false. Type: `BoolValue`
`width`	Specifies that the constraint applies only to shapes whose widths are less than the given width value. Type: `Value`, in user units
`interSpace`	Restricts the allowed spacing to integer multiples of this value. Use `withinRange` to limit this parameter to a specified range of values. Type: `Value`, in user units
`withinRange`	Specifies that the `interSpace` parameter applies only within this range. Type: `RangeValue`, in user units
`overLayer`	Specifies that the constraint applies only between the shapes that overlap a shape on this layer. Type: `LayerValue`

There are two exceptions that apply to this constraint and these are specified using the sets of optional parameters listed below.

Exception 1: The following three parameters collectively form an exception and all of them must be specified together. This exception applies only when all the conditions specified by these parameters are satisfied.

`parallelRunLength`	The constraint does not apply if the length of the violation is less than or equal to the specified length. Type: `Value`, in user units
`exceptOverLayer`	The constraint does not apply if the violation is over this layer. Type: `LayerValue`
`exceptRanges`	The constraint does not apply if the spacing falls within the specified range. Type: `RangeArrayValue`, in user units

Exception 2: The following four parameters collectively form an exception and must all be specified together. This exception applies only when all the conditions specified by these parameters are satisfied.

`numShapesRange`	The constraint does not apply if there are this many shapes on both sides of the violation with full parallel run length. Type: `RangeValue`
`maxWidth`	Specifies that shapes wider than this value do not count towards the `numShapesRange` value. Type: `Value`, in user units
`distance`	The constraint does not apply if the maximum distance between the shapes that form the exception is less than the specified distance. Type: `Value`, in user units
`exceptOtherRanges`	The constraint does not apply if the spacing falls within the specified range. Type: `RangeArrayValue`, in user units

Examples

- Example 1: oaAllowedSpacingRange with RangeArrayValue
- Example 2: oaAllowedSpacingRange with RangeArray1DTblValue

- Example 3: oaAllowedSpacingRange with RangeArray2DTblValue
- Example 4: oaAllowedSpacingRange with exceptOverLayer, parallelRunLength, and exceptRanges
- Example 5: oaAllowedSpacingRange with numShapesRange, maxWidth, distance, and exceptOtherRanges

Example 1: oaAllowedSpacingRange with RangeArrayValue

```
set_layer_constraint -constraint oaAllowedSpacingRange \
  -layer Metal2 -RangeArrayValue {"(0.1 0.19)" ">=0.4"}
```

Sets the permitted spacing ranges for `Metal2` shapes to greater than 0.1 and less than 0.19 user units, or greater than or equal to 0.4 user units.

Example 2: oaAllowedSpacingRange with RangeArray1DTblValue

```
set_layer_constraint -constraint oaAllowedSpacingRange -layer Metal1 \
  -RangeArray1DTblValue { \
      0.1 2 "(0.1 0.3)" ">= 0.4" \
      0.2 2 "(0.1 0.3)" ">= 0.5" \
      0.4 2 "(0.1 0.4)" ">= 0.6" }
```

Sets the allowed spacing ranges between `Metal1` shapes based on the width of one of the shapes, as follows:

- For width = 0.1, the allowed spacing range is greater than 0.1 and less than 0.3, or greater than or equal to 0.4.
- For width = 0.2, the allowed spacing range is greater than 0.1 and less than 0.3, or greater than or equal to 0.5.
- For width = 0.4, the allowed spacing range is greater than 0.1 and less than 0.4, or greater than or equal to 0.6.

Example 3: oaAllowedSpacingRange with RangeArray2DTblValue

```
set_layer_constraint -constraint oaAllowedSpacingRange -layer Metal1 \
  -TblCols {0.0 0.2 0.4}\
  -RangeArray2DTblValue \
      {0.0 {"(0.1 0.2)" "[0.3 0.4]" ">=0.5"} {} {} \
      0.2 {} {"(0.1 0.3)" ">=0.5"} {"(0.1 0.4]" ">=0.6"} \
      0.4 {} {"(0.1 0.4]" ">=0.6"} {"(0.1 0.5]" ">=0.7"}}
```

Sets the allowed spacing ranges between `Metal1` shapes based on the width of the two shapes, as follows:

- When both widths are equal to 0.0, the allowed spacing is greater than 0.1 and less than 0.2, or greater than or equal to 0.3 and less than or equal to 0.4, or greater than or equal to 0.5.
- When both widths are equal to 0.2, the allowed spacing is greater than 0.1 and less than 0.3, or greater than or equal to 0.5.
- When both widths are equal to 0.4, the allowed spacing is greater than 0.1 and less than or equal to 0.5, or greater than or equal to 0.7.
- When one width is equal to 0.2 and the other width is equal to 0.4, the allowed spacing is greater than 0.1 and less than or equal to 0.4, or greater than or equal to 0.6.
- The allowed spacing range is not specified for the following width combinations: (0.0, 0.2), (0.0, 0.4), (0.2, 0.0), and (0.4, 0.0).

For information about the Virtuoso technology file version of this constraint, see allowedSpacingRanges (one-layer) and allowedSpacingRanges (two-layer).

Example 4: oaAllowedSpacingRange with exceptOverLayer, parallelRunLength, and exceptRanges

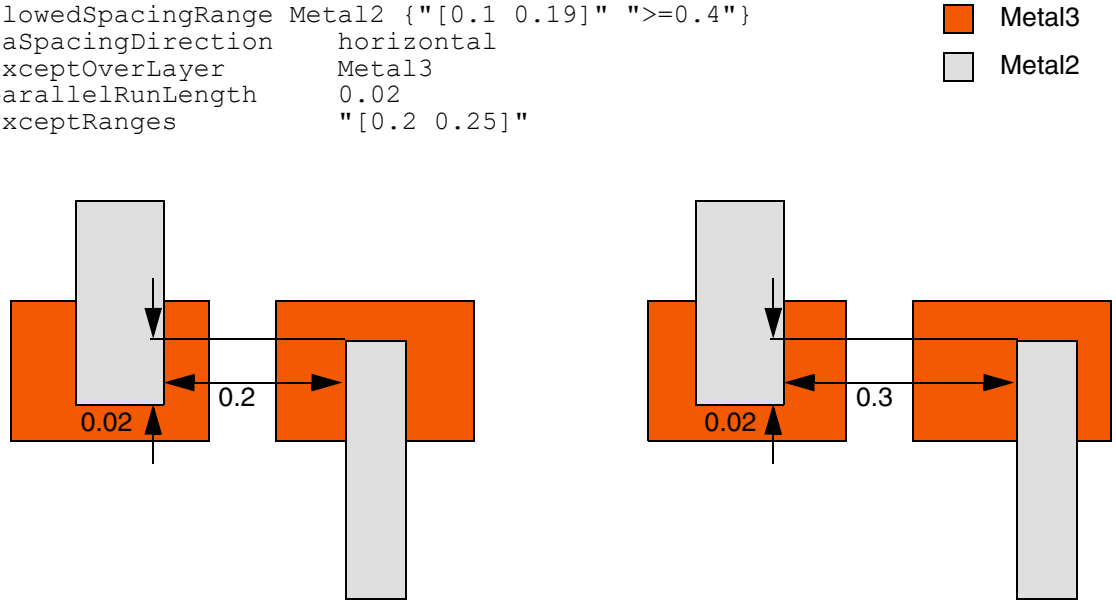
```
set_constraint_parameter -name exceptRanges -RangeArrayValue {[0.2 0.25]}
set_constraint_parameter -name parallelRunLength -Value 0.02
set_constraint_parameter -name exceptOverLayer -LayerValue Metal3
set_constraint_parameter -name oaSpacingDirection -StringAsIntValue \
  horizontalDirection
set_layer_constraint -constraint oaAllowedSpacingRange \
  -layer Metal2 -RangeArrayValue {"[0.1 0.19]" ">=0.4"}
```

Sets the allowed horizontal spacing between shapes on `Metal2` to be either greater than or equal to 0.1 and less than or equal to 0.19, or greater than or equal to 0.4, except when all of the following conditions are met:

- The violating `Metal2` shapes overlap `Metal3` shapes.
- The parallel run length between `Metal2` shapes is less than or equal to 0.02.
- The `Metal2` spacing is greater than or equal to 0.2 and less than or equal to 0.25.

Illustration for oaAllowedSpacingRange with exceptOverLayer, parallelRunLength, and exceptRanges

```
oaAllowedSpacingRange Metal2 {"[0.1 0.19]" ">=0.4"}
   oaSpacingDirection    horizontal
   exceptOverLayer       Metal3
   parallelRunLength     0.02
   exceptRanges          "[0.2 0.25]"
```



a) Constraint does not apply because the exception is triggered. The violating `Metal2` shapes are over `Metal3`, the PRL is 0.02 (<=0.02), and the spacing is 0.2 (>=0.2 and <=0.25).

b) FAIL. The spacing (0.3) is not in the allowed range (>=0.1 and <=0.19, or >=0.4) and does not meet the exception requirement (>=0.2 and <=0.25).

Example 5: oaAllowedSpacingRange with numShapesRange, maxWidth, distance, and exceptOtherRanges

```
set_constraint_parameter -name exceptOtherRanges -RangeArrayValue {[0.15 0.2]}
set_constraint_parameter -name distance -Value 0.08
set_constraint_parameter -name maxWidth -Value 0.2
set_constraint_parameter -name numShapesRange -RangeValue 3
set_constraint_parameter -name oaSpacingDirection -StringAsIntValue \
  horizontalDirection
set_layer_constraint -constraint oaAllowedSpacingRange \
  -layer Metal2 -RangeArrayValue {"[0.05 0.1]" ">=0.4"}
```

Sets the allowed `Metal2` horizontal spacing to be either greater than or equal to 0.05 and less than or equal to 0.1, or greater than or equal to 0.4, except when all of the following conditions are met:

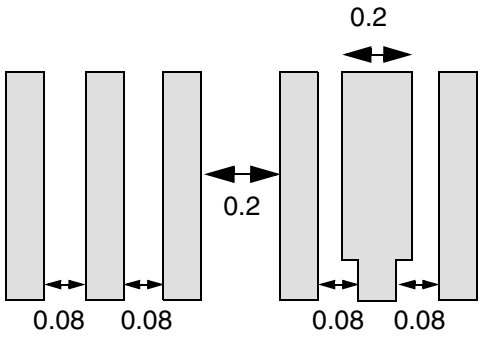
- There are three `Metal2` shapes more than 0.08 user units apart on both sides of the spacing violation with full parallel run length, each less than 0.2 user units wide.

- The spacing is greater than or equal to 0.15 and less than or equal to 0.2 user units.

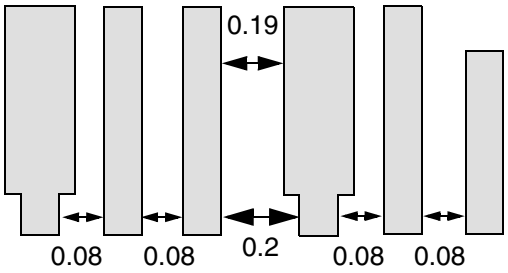
Illustration for oaAllowedSpacingRange with numShapesRange, maxWidth, distance, and exceptOtherRanges

```
oaAllowedSpacingRange Metal2 {"(0.05 0.1)" ">=0.4"}
   oaSpacingDirection    horizontal
   numShapesRange        3
   maxWidth              0.2
   distance              0.08
   exceptOtherRanges     "[0.15 0.2]"
```

Metal2



a) The constraint does not apply because the exception is triggered. There are three `Metal2` shapes on either side, each at most 0.2 user units wide. These shapes are <=0.08 apart and the spacing (0.2) is within the exception range of >=0.15 and <=0.2.



b) FAIL. The exception does not apply because three shapes along the full parallel run length are needed on either side. The parallel run length of the rightmost shape is not equal to that of the other shapes.

Related Topics

Spacing Constraints

check_space

oaMinEndOfNotchSpacing

Specifies the minimum spacing between a notch and shapes overlapping within the extent of the notch.

oaMinEndOfNotchSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

The `oaMinEndOfNotchSpacing` constraint has a Value to represent the minimum required spacing, in user units.

Format	Example

Tcl	`set_constraint_parameter -name oaNotchWidth -Value f_notchWidth`
	`set_constraint_parameter -name oaNotchSpacing -Value f_notchSpacing`
	`set_constraint_parameter -name oaNotchLength -Value f_notchLength`
	`set_layer_constraint -constraint oaMinEndOfNotchSpacing \`
	`-layer s_layer -Value f_endOfNotchSpacing`

LEF	`LAYER s_layer`
	`TYPE ROUTING ;`
	`SPACING f_endOfNotchSpacing ENDOFNOTCHWIDTH f_notchWidth`
	`NOTCHSPACING f_notchSpacing NOTCHLENGTH f_notchLength ;`

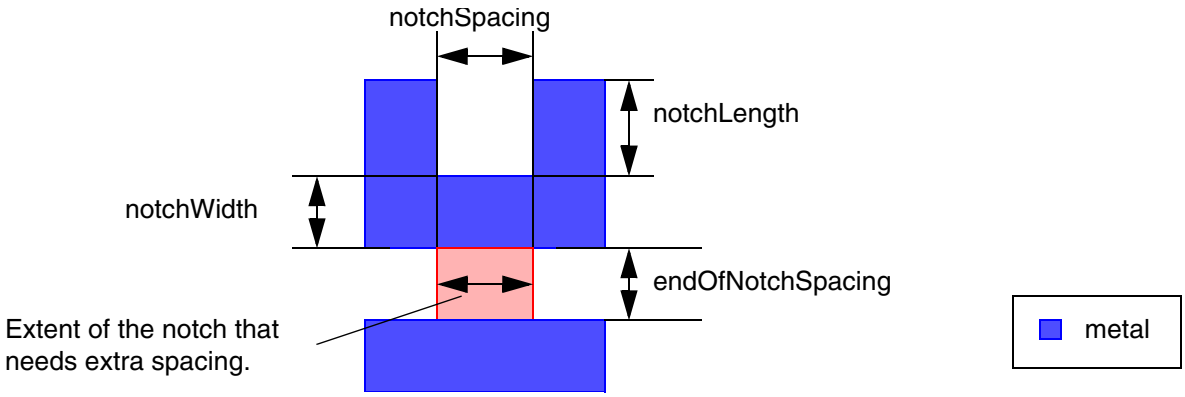
Virtuoso	`spacings(`
	`( minNotchSpacing s_layer`
	`'notchWidth f_notchWidth 'notchLength f_notchLength`
	`'notchSpacing f_notchSpacing f_endOfNotchSpacing)`
	`)`

Parameters

The following parameters determine when the constraint applies:

- `oaNotchWidth` (Value) specifies that the constraint only applies when the width of a notch is less than this value, in user units.

- `oaNotchSpacing` (`Value`) specifies that the constraint applies only when the notch spacing is less than or equal to this value, in user units.
- `oaNotchLength` (`Value`) specifies that the constraint applies only when the length of a notch is greater than or equal to this value, in user units.



Examples

In this example, the notch metal at the bottom end of a U-shaped notch must have spacing that is greater than or equal to 0.14 μm, if the notch metal has a width that is less than 0.15 μm, notch spacing that is less than or equal to 0.16 μm, and notch length that is greater than or equal to 0.08 μm. See the following figure for different layout examples for these rules.

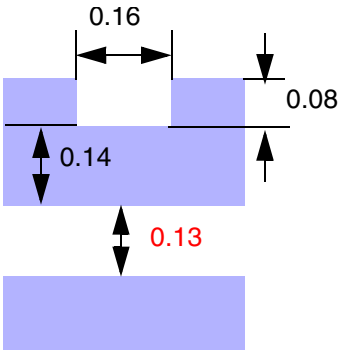
Format	Example

Tcl	`set_layer_constraint -constraint minSpacing -layer Metal1 -Value 0.10`
	`set_constraint_parameter -name oaNotchWidth -Value 0.15`
	`set_constraint_parameter -name oaNotchSpacing -Value 0.16`
	`set_constraint_parameter -name oaNotchLength -Value 0.08`
	`set_layer_constraint -constraint oaMinEndOfNotchSpacing \`
	`-layer Metal1 -Value 0.14`

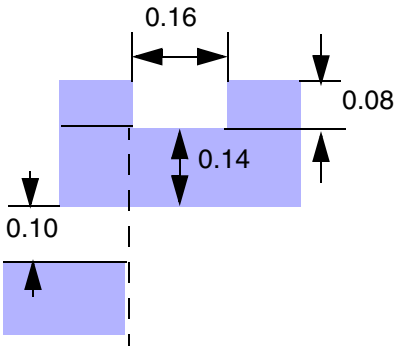
LEF	`LAYER Metal1`
	`TYPE ROUTING ;`
	`SPACING 0.10 ; #default spacing`
	`SPACING 0.14 ENDOFNOTCHWIDTH 0.15 NOTCHSPACING 0.16 NOTCHLENGTH 0.08`
	`;`

Virtuoso	`spacings(`
	`( minSpacing Metal1 0.10 )`
	`( minNotchSpacing Metal1`
	`'notchWidth 0.15 'notchLength 0.08`
	`'notchSpacing 0.16 0.14)`
	`)`

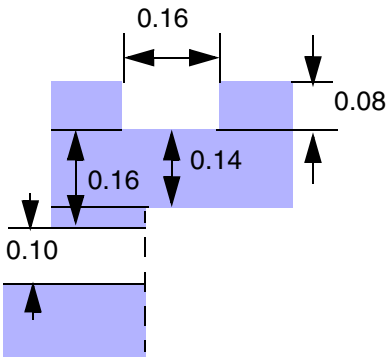
oaMinEndOfNotchSpacing Constraint Examples



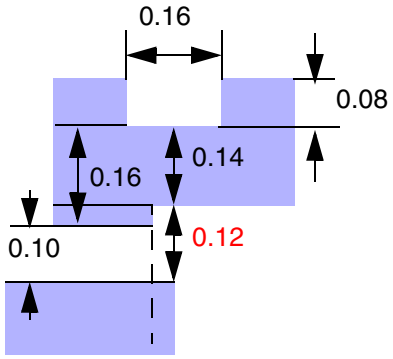
Violation because at least 0.14 μm is required at the end of notch.



No violation because neighboring shape is not within extent of notch. The default spacing of 0.10 μm is required.



No violation because notch metal width is >= 0.15 for the overlap, therefore extra spacing is not required.



Violation because notch metal width is < 0.15 for part of the overlap, therefore 0.14 μm spacing is required.

Related Topics

Spacing Constraints

oaMinNotchSpacing

Lets you base the notch spacing on the length of the short side of a notch. It sets the minimum spacing, edge-to-edge, for a single-layer, merged shape when the length of the notch is less than the specified length.

oaMinNotchSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

oaMinNotchSpacing constraints have a Value that represents the minimum notch spacing, in user units.

Format	Example

Tcl	`set_constraint_parameter -name oaNotchLength -Value f_notchLength`
	`set_layer_constraint -constraint oaMinNotchSpacing \`
	`-layer s_layer -Value f_notchSpacing`

LEF	`LAYER s_layer`
	`TYPE ROUTING ;`
	`SPACING f_notchSpacing NOTCHLENGTH f_notchLength ;`

Virtuoso	`spacings(`
	`( minNotchSpacing s_layer 'notchLength f_notchLength f_space`
	`)`
	`)`

Parameters

- `oaNotchLength` (Value) specifies that the constraint applies only when the length of the short side of a notch is less than this value, in user units.
- `betweenConcaveCorners` (Value) specifies that the constraint only applies to notches where the shortest notch length is between two concave corners.

- `width` (`Value`) is an optional parameter which specifies that the minimum notch length spacing only applies if the width of a single side of the notch is greater than or equal to `width`.

Examples

Sets the minimum notch spacing to 0.12 for notches having a length less than `0.15`, while the minimum spacing is `0.10`. See the following figure for examples using this rule.

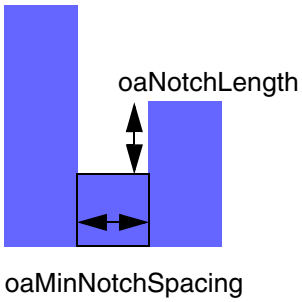
Format	Example

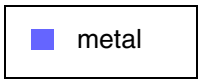
Tcl	`set_layer_constraint -constraint minSpacing 0.10 -layer Metal1`
	`set_constraint_parameter -name oaNotchLength -Value 0.15`
	`set_layer_constraint -constraint oaMinNotchSpacing \`
	`-layer Metal1 -Value 0.12`

LEF	`LAYER Metal1`
	`TYPE ROUTING ;`
	`SPACING 0.10 ; #default spacing`
	`SPACING 0.12 NOTCHLENGTH 0.15 ;`

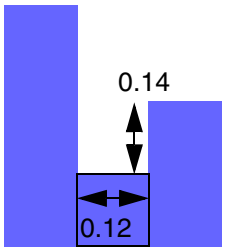
Virtuoso	`spacings(`
	`( minSpacing Metal1 0.10 )`
	`( minNotchSpacing Metal1 'notchLength 0.15 0.12 )`
	`)`

oaMinNotchSpacing Examples

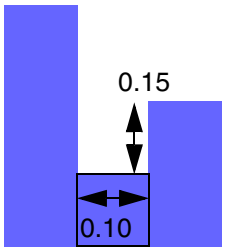




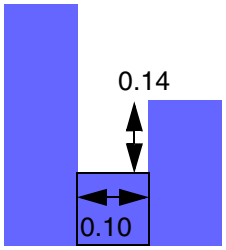
When the shorter notch length is less than `oaNotchLength`, the notch spacing, edge to edge, on the specified layer must be greater than or equal to the `oaMinNotchSpacing` constraint.



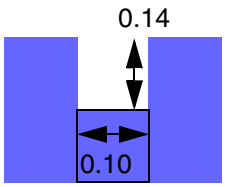
a) OK; meets constraint requirements



b) OK; short side notch edge is not less than 0.15, so constraint does not apply



c) Violation; constraint applies because short end of notch is less than 0.15 and notch spacing is less than oaMinNotchSpacing.



d) Violation; same as case c).

Related Topics

Spacing Constraints

shapeRequiredBetweenSpacing

Specifies the minimum spacing between shapes on layer1 that do not require a shape on layer2 between them. Optionally, this constraint depends on the direction of the spacing between the layer1 shapes, either horizontally or vertically.

shapeRequiredBetweenSpacing Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing

Value Type

`Value`	Specifies the distance, in user units, between layer1 shapes. layer1 shapes closer than this distance must have a layer2 shape between them.

Optional Parameter

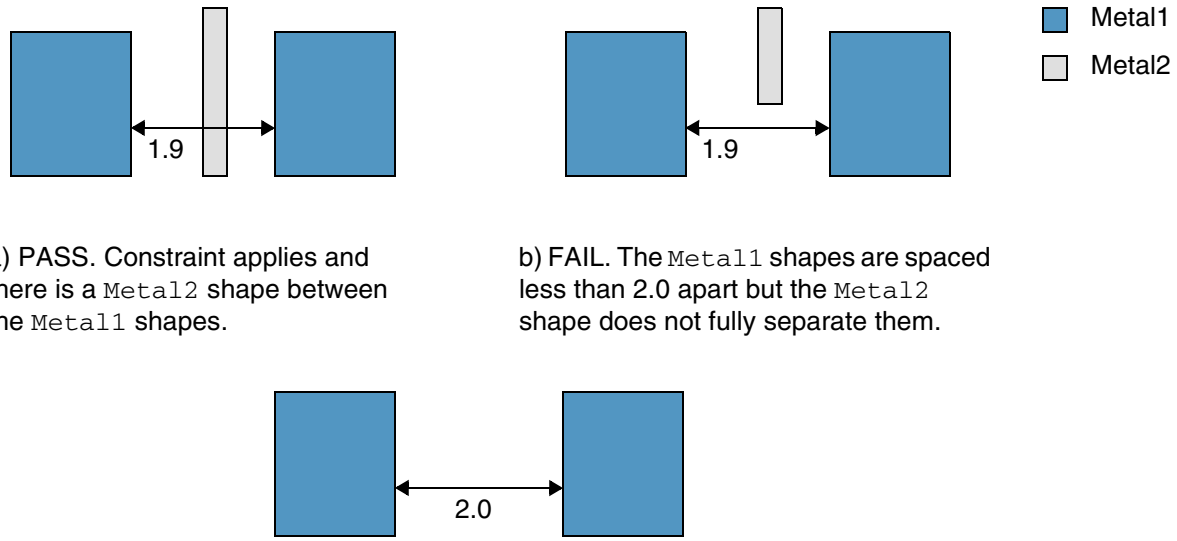
`oaSpacingDirection`	Specifies the measurement direction for the layer1 distance. Type: `IntValue` 0 any 1 horizontal 2 vertical

Examples

Specifies that `Metal1` shapes that are closer than 2.0 user units, when measured in the horizontal direction, must have a `Metal2` shape between them.

```
set_constraint_parameter -name oaSpacingDirection -IntValue 1
set_layerpair_constraint -constraint shapeRequiredBetweenSpacing \
  -layer1 Metal1 -layer2 Metal2 -Value 2.0
```

Illustration for shapeRequiredBetweenSpacing



a) PASS. Constraint applies and there is a `Metal2` shape between the `Metal1` shapes.

b) FAIL. The `Metal1` shapes are spaced less than 2.0 apart but the `Metal2` shape does not fully separate them.

c) PASS. The `Metal1` shapes have a horizontal spacing of 2.0 user units and do not need a `Metal2` shape between them.

Related Topics

Spacing Constraints

trimMinSpacing

Defines the minimum spacing between shapes on the specified trim metal layer.

trimMinSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	design, foundry
Category	Spacing
Group Operators	AND, OR

Value Type

Value	Specifies the minimum spacing, edge-to-edge, between shapes on the specified layer.

Required Parameters

None

Optional Parameter

prlSpacing	Specifies that the spacing between two shapes must be greater than or equal to the first given value if the parallel run length between the shapes is zero; if the parallel run length between the shapes is greater than zero, the spacing between them must be greater than or equal to the second given value. Otherwise, the spacing between the two shapes must be greater than or equal to the constraint value. Type: DualValue

`endToEndSpacing`

Specifies that end-to-end spacing in the given direction must be greater than or equal to this value if the parallel run length between the shapes is greater than `parallelRunLength`, when specified, or equal to zero.

Type: `Value`

`oaSpacingDirection`

Specifies the direction in which end-to-end spacing is measured.

Valid Values:

`anyDirection`	`0`	Horizontally and vertically (default)
`horizontalDirection`	`1`	Horizontally only
`verticalDirection`	`2`	Vertically only

Type: `IntValue`, `StringAsIntValue`

`parallelRunLength`

Specifies that end-to-end spacing is applied only if the parallel run length between the shapes is greater than this value, when specified.

Type: `Value`

`exactAligned`

Specifies that end-to-end spacing must be greater than or equal to this value if the shapes are aligned.

If `parallelRunLength` is not specified, end-to-end spacing is applied in Euclidian measurement. If `parallelRunLength` is specified, end-to-end spacing is applied in Manhattan measurement.

Type: `Value`

`exceptWidth`

Specifies that if a shape on `layer` is between two trim shapes and has width greater than or equal to this value, all end-to-end spacing rules are exempted.

Type: `Value`

`layer`

Specifies the layer on which `exceptWidth` applies. This is the layer that the trim metal trims.

Type: `LayerValue`

`sameMask`

Specifies that the constraint applies only between shapes on the same mask.

Type: `BoolValue`

Examples

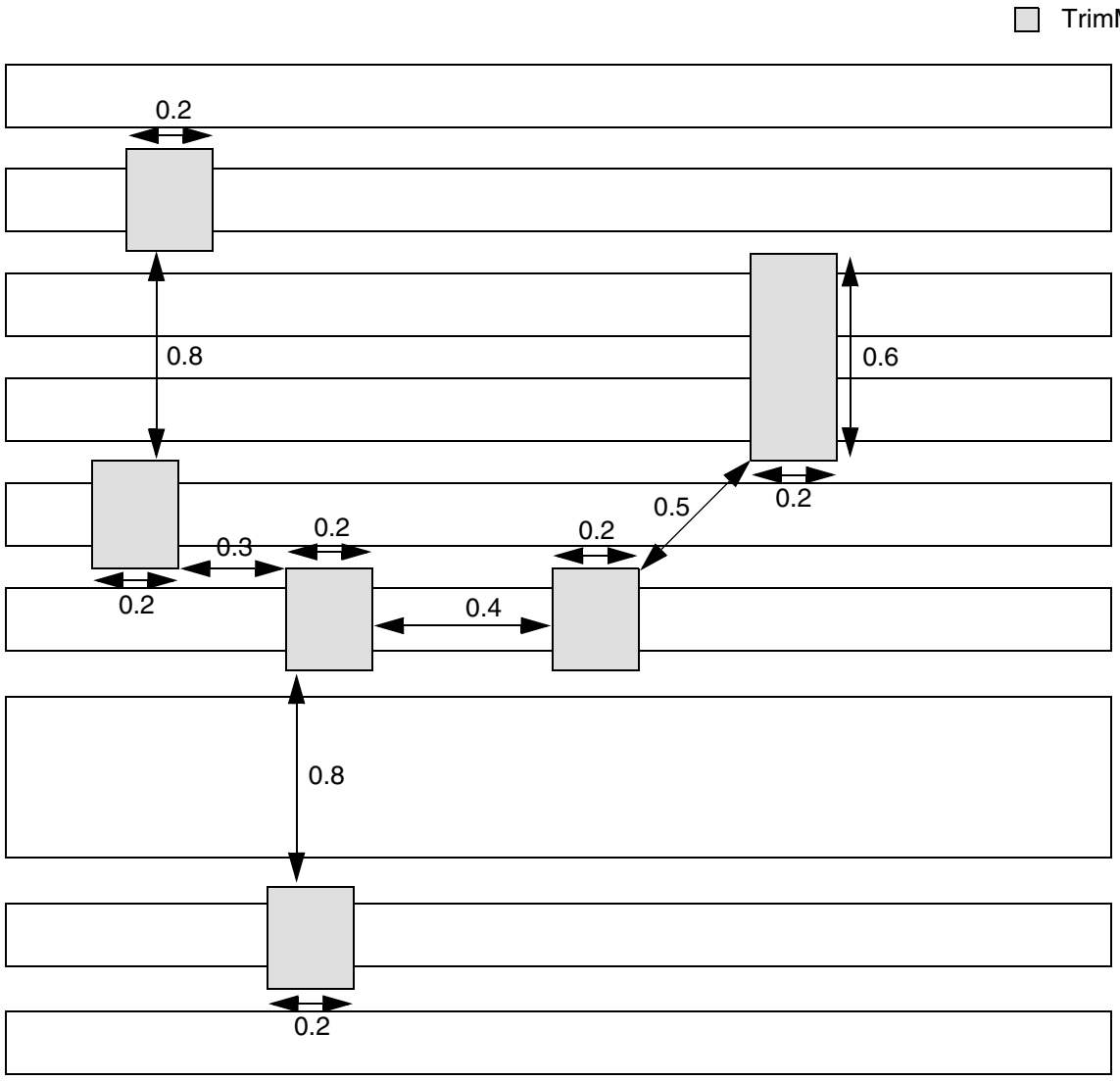
- Example 1: trimMinSpacing with prlSpacing, endToEndSpacing, and verticalDirection
- Example 2: trimMinSpacing with endToEndSpacing, verticalDirection, parallelRunLength, and exactAligned
- Example 3: trimMinSpacing with endToEndSpacing, vertical, parallelRunLength, exceptWidth, and layer

Example 1: trimMinSpacing with prlSpacing, endToEndSpacing, and verticalDirection

```
set_constraint_parameter -name prlSpacing -DualValue {0.3 0.4}
set_constraint_parameter -name endToEndSpacing -Value 0.8
set_constraint_parameter -name oaSpacingDirection -IntValue 2
set_layer_constraint -constraint trimMinSpacing -layer TrimMetal1 -Value 0.5
```

Specifies that the spacing between two shapes must be greater than or equal to 0.3 if the parallel run length between the two shapes is zero. If the parallel run length between the two shapes is greater than zero, the spacing between them must be greater than or equal to 0.4. Otherwise, the spacing between the two shapes must be greater than or equal to 0.5.

The end-to-end spacing in the vertical direction must be greater than or equal to 0.8 if the parallel run length between the shapes is greater than zero.

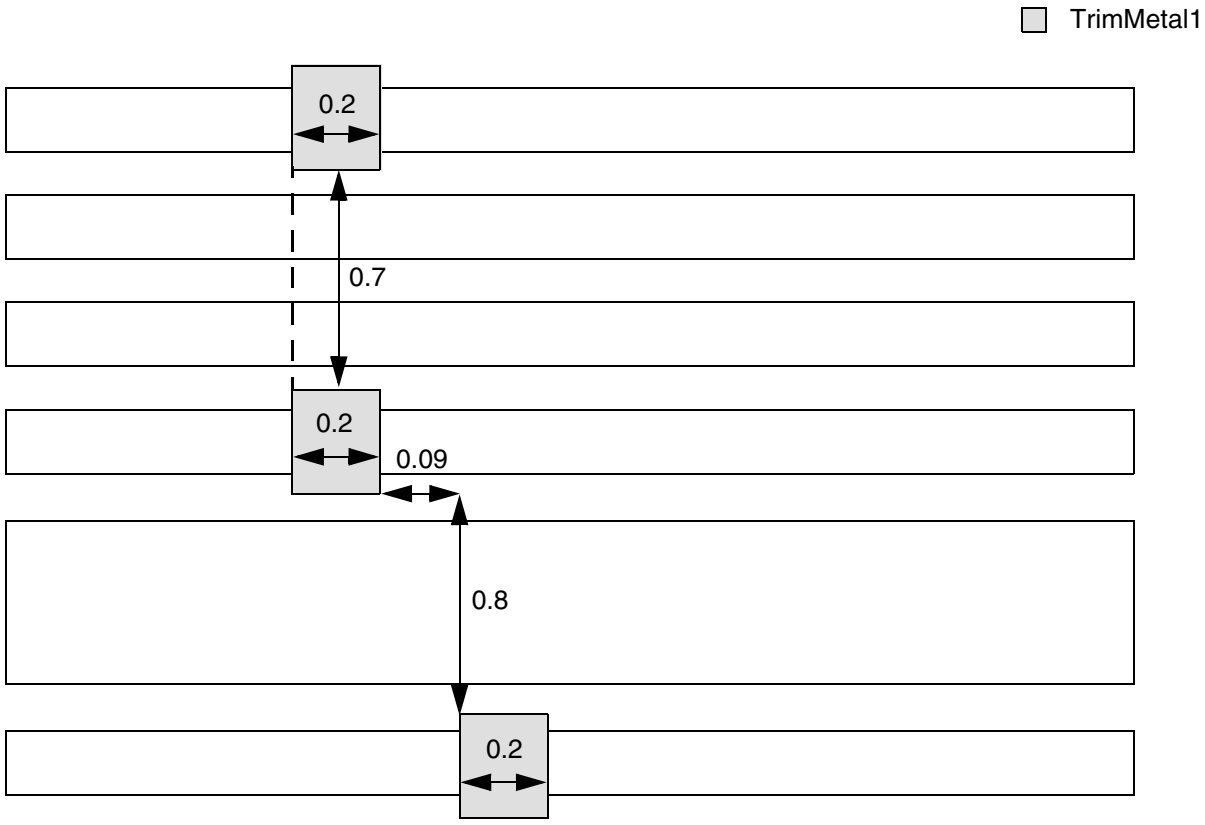


PASS. The width of all TrimMetal1 shapes is equal to 0.2 and the maximum length is 0.6. The spacing is 0.3 when the parallel run length is zero and 0.4 when the parallel run length is greater than zero. The spacing between shapes that have parallel run length less than zero is 0.5. The vertical end-to-end spacing between shapes with parallel run length greater than zero is 0.8.

Example 2: trimMinSpacing with endToEndSpacing, verticalDirection, parallelRunLength, and exactAligned

```
set_constraint_parameter -name endToEndSpacing -Value 0.8
set_constraint_parameter -name oaSpacingDirection -IntValue 2
set_constraint_parameter -name parallelRunLength -Value -0.1
set_constraint_parameter -name exactAligned -Value 0.7
set_layer_constraint -constraint trimMinSpacing -layer TrimMetal1 -Value 0.5
```

Specifies that the end-to-end spacing in the vertical direction must be greater than or equal to 0.8 if the parallel run length between the shapes is greater than -0.1. If the shapes are aligned, the end-to-end spacing in the vertical direction must be greater than or equal to 0.7. Otherwise, the spacing between the shapes must be at least 0.5.

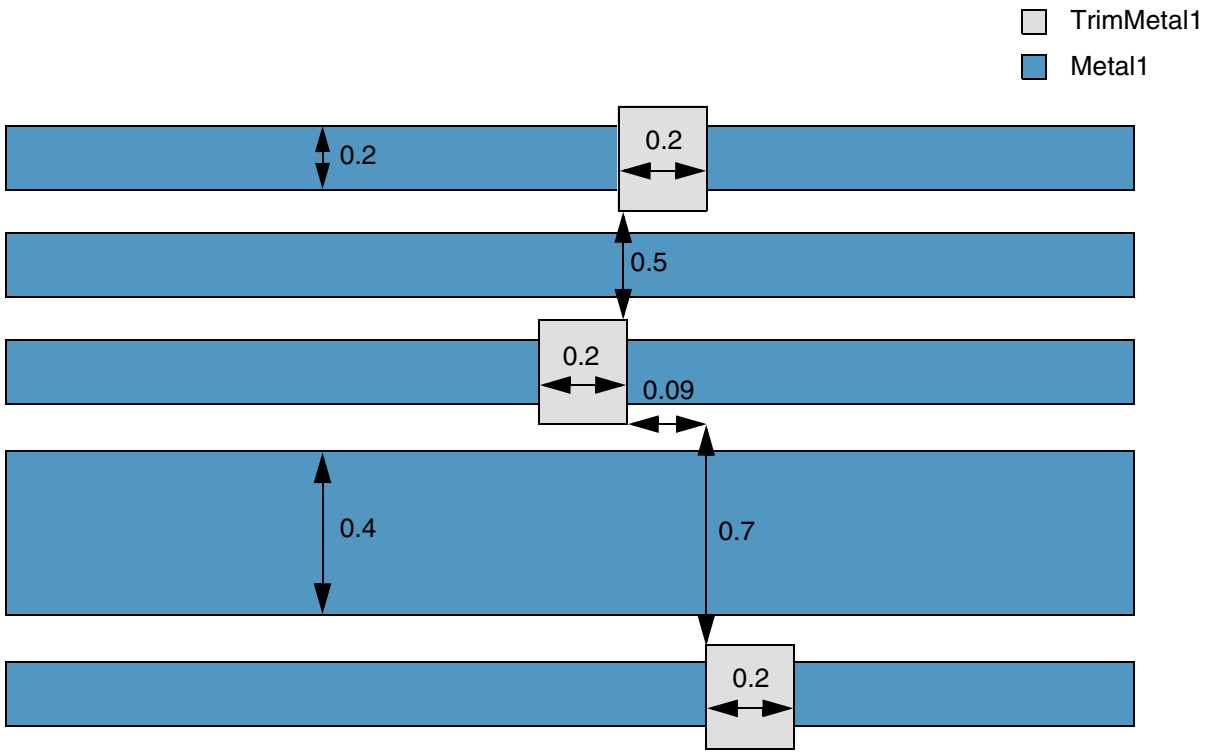


PASS. The vertical end-to-end spacing between shapes with parallel run length equal to 0.09 (<0.1) is 0.8, and the vertical end-to-end spacing between shapes that are aligned is 0.7.

Example 3: trimMinSpacing with endToEndSpacing, vertical, parallelRunLength, exceptWidth, and layer

```
set_constraint_parameter -name endToEndSpacing -Value 0.8
set_constraint_parameter -name oaSpacingDirection -IntValue 2
set_constraint_parameter -name parallelRunLength -Value -0.1
set_constraint_parameter -name exceptWidth -Value 0.4
set_constraint_parameter -name layer -LayerValue Metal1
set_layer_constraint -constraint trimMinSpacing -layer TrimMetal1 -Value 0.5
```

Specifies that the end-to-end spacing in the vertical direction must be greater than or equal to 0.8 if the parallel run length between the shapes is greater than -0.1. If the width of a shape on Metal1 is greater than or equal to 0.4, the end-to-end spacing requirement is exempted. Otherwise, the spacing between the shapes must be at least 0.5.



FAIL. The vertical end-to-end spacing between the middle and top shapes, which have parallel run length greater than -0.1, is 0.5 (<0.8). The vertical end-to-end spacing between the middle and bottom shapes (0.7<0.8) is exempted because the width of the intermediate Metal1 shape is 0.4.

Related Topics

Spacing Constraints

check_space

17

Via Construction Constraints

This topic lists the via construction constraints.

allowedCutClass
cutClass
forbiddenCutClassSpacingRange
maxStressLength
minAdjacentViaSpacing
minCutClassClearance
minCutClassSpacing
minLargeViaArrayCutSpacing
minLargeViaArraySpacing
minLargeViaArrayWidth
minNeighborViaSpacing
minParallelViaSpacing
minParallelWithinViaSpacing
minRedundantViaSetback
minSameMetalSharedEdgeViaSpacing
minViaExtension
oaMinOrthogonalViaSpacing
oaMinParallelViaClearance
oaMinViaClearance
oaMinViaSpacing
rectangularLargeViaArraysAllowed
sameNetLargeViaSpacing
viaBarAdjacentSpacing
viaEdgeType
viaStackingAllowed
viaStackLimit

Related Topics

Tcl Constraint Commands

Manage Constraints

allowedCutClass

Specifies whether a cut class on the given cut layer is allowed between shapes of certain widths. If `lowerLayer` and `upperLayer` are not specified, the `viaDefs` for the cut layer are used to determine the lower and upper layers. The lower and upper direction is determined by the wider dimension of the shape.

Note: If all cut classes are allowed on all wire width combinations, the `allowedCutClass` constraint need not be defined.

allowedCutClass Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction
Group Operators	AND, OR

Value Type

`Int2DTblValue`	Specifies whether a cut class on the given cut layer is allowed between shapes of certain widths. The column (`-TblCols`) and row headers for the 2-D table specify the widths of the shapes on the lower and upper layers, respectively. The value is set to `0` if the specified cut class is disallowed for the corresponding shape width combination, and `1` if allowed.

Required Parameter

`cutClass`	Specifies the cut class width and length. Type: `DualValue`

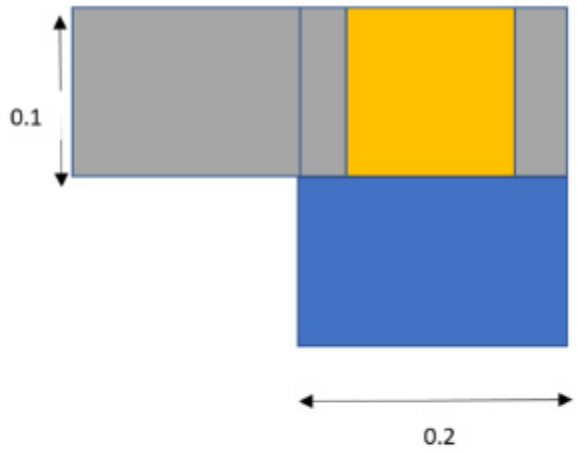
Optional Parameters

`lowerLayer`	The metal layer below the cut layer. If not specified, the default stack is used to determine the lower metal layer. Type: `LayerValue`
`upperLayer`	The metal layer above the cut layer. If not specified, the default stack is used to determine the upper metal layer. Type: `LayerValue`
`lowerDirection`	The direction of the shape on the lower layer. The direction is determined by the wider dimension of the shape. The constraint applies if the shape has the specified direction. The values that can be specified for the lower direction are: ■ `0`: `anyLowerDir` ■ `1`: `horizontalLowerDir` ■ `2`: `verticalLowerDir` Default is `0`, which means any lower direction. Type: `IntValue`
`upperDirection`	The direction of the shape on the upper layer. The direction is determined by the wider dimension of the shape. The constraint applies if the shape has the specified direction. The values that can be specified for the upper direction are: ■ `0`: `anyUpperDir` ■ `1`: `horizontalUpperDir` ■ `2`: `verticalUpperDir` Default is `0`, which means any upper direction. Type: `IntValue`

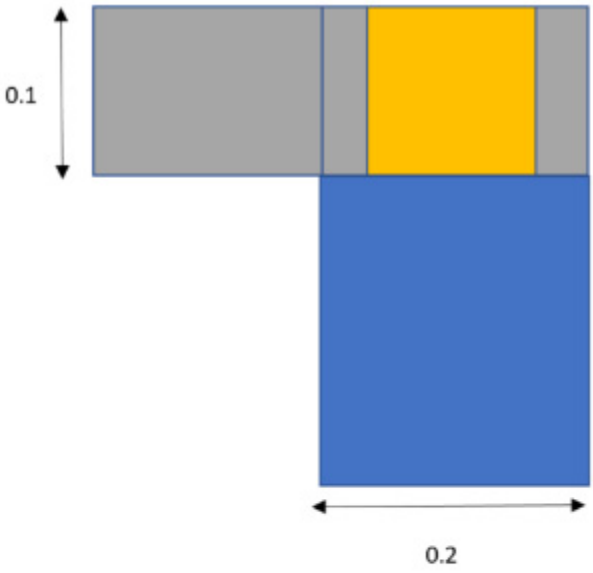
Examples

```
set_constraint_parameter -name cutClass -DualValue {0.1 0.1}
set_constraint_parameter -name lowerLayer -LayerValue Metal1
set_constraint_parameter -name upperLayer -LayerValue Metal2
set_constraint_parameter -name lowerDirection -IntValue 1
set_constraint_parameter -name upperDirection -IntValue 1
set_layer_constraint -constraint allowedCutClass \
-layer Via1 -TblCols {0.1 0.2} -Int2DTblValue {0.1 1 0 0.2 0 1}
```

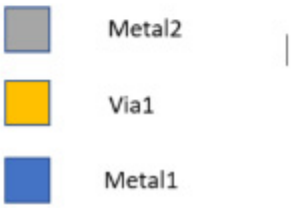
Specifies that a 0.1x0.1 cut class on layer Via1 is allowed between Metal1 and Metal2 wires only if both wires are greater than 0.1 or 0.2 user units in width and direction for both Metal1 and Metal2 wire is horizontal.



PASS. For wires with width 0.1 and 0.2, Via1 Cut Class(size 0.1x0.1) is allowed only if direction of both wires is 1 i.e horizontal.



FAIL. For wires with width 0.1 and 0.2, Via1 Cut Class(size 0.1x0.1) is allowed only if direction of both wires is 1 i.e horizontal. The direction of Metal1 wire is vertical in this case.



Related Topics

Via Construction Constraints

cutClass

Defines a cut class name and specifies the via width and via length, in user units, for cuts that belong to the given class.

Cut classes are referenced by other constraints, implicitly by class size, to scope those constraints to cuts of the specific class. In cases where a constraint for a cut layer is scoped to a cut class, the rest of the constraints of that type for that layer must also reference a cut class. There is no fallback for a constraint that does not reference a cut class for any cut sizes not covered by cut class-specific constraints.

cutClass Quick Reference

Constraint Type	Layer
Value Types	`DualValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction
Group Operators	OR

Value Type

`DualValue`	Specifies the exact width (viaWidth) and length (viaLength), in user units, of a cut that should belong to this cut class. Multiple cut classes for a given cut layer should be collected as multiple cutClass constraints in an OR constraint group.

Required Parameter

`className`	Specifies the cut class name, which is used to reference the cut class by other constraints. Type: `StringValue`

Optional Parameters

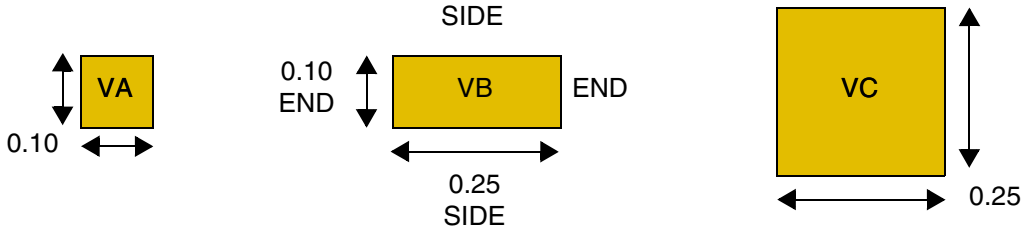
`numCuts`	Specifies the equivalent number of cuts of the cut class for the minimum cut rule and calculation of resistance value. If a minimum cut rule requires *n* cuts, having round up of *n*/ `numCuts` vias of this cut class is sufficient. Resistance value for this cut class vias would be *resistance of the cut layer*/ `numCuts`. The default is 1. Type: `IntValue`
`fixedOrientation`	Specifies that the first value of the dual value is always interpreted as the x-span of the cut class, and the second value is always interpreted as the y-span of the cut class. Type: `BoolValue`

Examples

```
create_constraint_group -name cutClassgp -opType or -db tech
add_constraint_group -subGroupName cutClassgp -groupType foundry

set_constraint_parameter -name className -StringValue VA
set_constraint_parameter -name numCuts -IntValue 1
set_layer_constraint -constraint cutClass -layer Via1 -create true \
  -group cutClassgp -DualValue {0.10 0.10}

set_constraint_parameter -name className -StringValue VB
set_constraint_parameter -name numCuts -IntValue 2
set_layer_constraint -constraint cutClass -layer Via1 -create true \
  -group cutClassgp -DualValue {0.10 0.25}

set_constraint_parameter -name className -StringValue VC
set_constraint_parameter -name numCuts -IntValue 4
set_layer_constraint -constraint cutClass -layer Via1 -create true \
  -group cutClassgp -DualValue {0.25 0.25}
```

Creates three cut classes named VA, VB, and VC for layer Via1. VA and VC are square cut classes (0.10 user units^2 and 0.25 user units^2, respectively). VB is a rectangular cut class of 0.10 x 0.25. The VB shorter side (0.10) is called the *end* and the longer side (0.25) is considered the *side*, as shown in the following figure.

Large Via Examples for cutClass



Related Topics

Via Construction Constraints

forbiddenCutClassSpacingRange

Restricts the spacing between two shapes of a given cut class on the given layer. This constraint can optionally apply only for the spacing between short (or end) edges of a rectangular cut class with a common parallel run length greater than zero.

forbiddenCutClassSpacingRange Quick Reference

Constraint Type	Layer
Value Types	`RangeValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Types

`RangeValue`	Specifies the spacing between cut shapes that is considered illegal. Any spacing that falls within the range is forbidden. For example, if `RangeValue` is `"[`*minSpacing maxSpacing*`]"`, then any spacing where *minSpacing* <= spacing <= *maxSpacing* is illegal. For this example, spacing between the two shapes must be less than *minSpacing* or greater than *maxSpacing*.

Required Parameter

`cutClass`	Specifies that the constraint applies only to cut shapes of the specified cut class. Type: `DualValue`

Optional Parameter

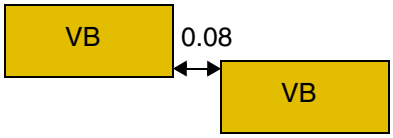
`shortEdgeOnly`	Specifies that the constraint only applies between short sides of the cut class shapes. If this parameter is specified, the specified cutClass should be a rectangular cut class. Default is `false`. Type: `BoolValue`

Examples

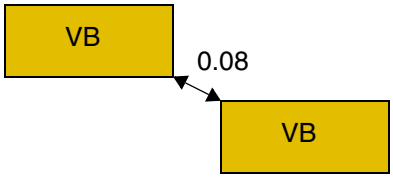
```
set_constraint_parameter -name cutClass -DualValue {0.3 0.4}
set_constraint_parameter -name shortEdgeOnly -BoolValue true
set_layer_constraint -constraint forbiddenCutClassSpacingRange \
  -layer v1 -RangeValue {"[0.7 0.9]"}
```

Specifies that the spacing between two end edges of `VB` (0.3x0.4) rectangular cut shapes on cut layer `v1` with parallel run length greater than 0 must not be greater than or equal to 0.07 or less than or equal to 0.09 user units.

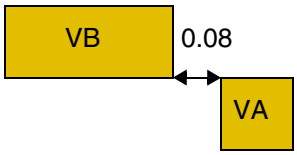
Illustration for forbiddenCutClassSpacingRange



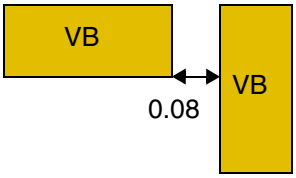
a) FAIL. The two rectangular cuts have common parallel run length > 0, but the spacing between the two short/end edges is inside the forbidden spacing range.



b) PASS. The two rectangular cuts do not have a common parallel run length > 0, so the constraint does not apply. If the `shortEdgeOnly` parameter was omitted, this would be a violation.



c) Constraint does not apply. It applies only to spacing between two VB cuts.



d) Constraint does not apply. It applies only between two short/end VB edges. It would be a violation if the `shortEdgeOnly` parameter was omitted.

Related Topics

Via Construction Constraints

check_space

maxStressLength

Specifies the maximum distance that wire can extend from a cut.

maxStressLength Quick Reference

Constraint Type	Layer pair
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`maxStressLength` constraints have a `Value` that specifies the maximum distance that wire can extend from a cut.

Related Topics

Via Construction Constraints

minAdjacentViaSpacing

Specifies the required minimum distance in user units between adjacent via cuts. A via cut is considered adjacent if it is within the specified distance from another cut in any direction including a 45-degree angle. Additional distance is required when multiple cuts are adjacent to ensure that geometries are not merged during fabrication.

Optional parameters determine whether via spacing is measured center-to-center or edge-to-edge, whether the constraint applies to only certain connectivity types, and whether the constraint applies to vias on power and ground nets. In some processes, a specific spacing rule may only apply if the vias have no parallel run length, as opposed to being aligned.

minAdjacentViaSpacing Quick Reference

Constraint Type	Layer
Value Types	Value
Database Types	Design, Technology
Scope	net, route, Term, instTerm, blockage, net group, group2group, reflexive, transreflexive, design, foundry
Category	Via Construction
Group Operators	AND, OR

Value Type

Value	Specifies the minimum distance required between adjacent vias.

Required Parameters

distance	Specifies the maximum distance apart, in user units, for cuts to be considered adjacent. Type: Value
numCuts	Specifies the minimum required number of adjacent cuts for this constraint to apply. Type: IntValue

The constraint applies when at least `numCuts` other vias are adjacent within a given `distance`.

Optional Parameters

`cutClass`	The constraint applies only to vias with the given dimensions. Type: `DualValue`
`exactAligned`	The constraint applies when the cut has this number of via cuts that are perfectly aligned horizontally and/or vertically and are less than `distance` from each other. Otherwise, the constraint applies when there are at least `numCuts` non-perfectly aligned cuts. The `numCuts` value must be smaller than the `exactAligned` value. Type: `IntValue`
`exactSpacing`	Specifies whether the `minAdjacentViaSpacing` value is an exact value. This is used for contacts that must be aligned. This parameter is not supported by LEF. Type: `BoolValue`
`oaCenterToCenter`	Determines whether the minimum spacing applies center-to-center (`true`) or edge-to-edge (`false`, default). Type: `BoolValue`
`oaConnectivityType`	Specifies the shapes to which this constraint applies, based on their connectivity. Type: `StringAsIntValue`
`oaExceptSamePGNet`	Specifies that the constraint does not apply to power and ground nets. The default is `false`, meaning the constraint applies to all nets. Type: `BoolValue`
`noParallelRunLength`	Specifies that the rule applies only among cuts with no common parallel run length. Default is false. Type: `BoolValue`
`allCuts`	Specifies that a neighbor cut would belong to any cut classes. Default is false. Type: `BoolValue`

`cutSizeRangeArray` Specifies ranges for the xSpan and ySpan values of cuts.

Type: RangeArrayValue

- `{xRange yRange}`
  Cuts with xSpan values that fall within the `xRange` and ySpan values that fall within the `yRange` for all cuts.
- `{xRange1 yRange1 xRange2 yRange2}`
  First cut with xSpan values in `xRange1` and ySpan in `yRange1` and neighboring cuts with xSpan in `xRange2` and ySpan in `yRange2`.

`twoCuts` Specifies that the via spacing applies between two cuts only if each of them has at least `twoCuts` neighbor cuts at less than the `distance`.

Type: IntValue

`exceptOppositeCornerNeighbors`

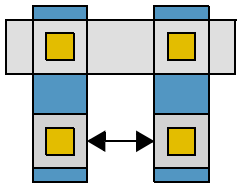
Specifies that the constraint does not apply if all neighbors are on opposite corners.

Type: BoolValue

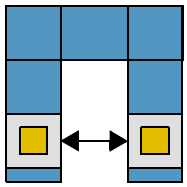
oaConnectivityType Parameter Values for Via Spacing Constraints

String	Integer Value	Constraint applies to:
`anyConnectivity`	0	(Default) Any connectivity
`sameNetConnectivity`	1	Same net only
`sameIslandConnectivity`	2	Via cuts connected by contiguous same metal
`directConnectShapesConnectivity`	3	Via cuts directly connected by the same metal
`sameViaConnectivity`	4	Via cuts with the same metal above and same metal below

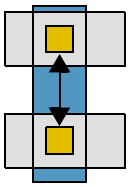
Connectivity Type Examples for Vias



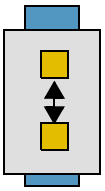
Same net applies.



Same island and **same net** apply.



Direct connect, **same island** and **same net** apply.



Same via, direct connect, **same island** and **same net** apply.

Examples

- Example 1: minAdjacentViaSpacing Edge-to-edge
- Example 2: minAdjacentViaSpacing Center-to-Center
- Example 3: minAdjacentViaSpacing with noParallelRunLength

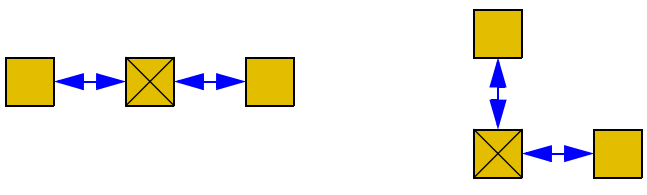
Example 1: minAdjacentViaSpacing Edge-to-edge

```
set_constraint_parameter -name numCuts -IntValue 3
set_constraint_parameter -name distance -Value 0.1
set_constraint_parameter -name oaCenterToCenter -BoolValue false
set_layer_constraint -constraint minAdjacentViaSpacing -layer VIA1 -Value 0.11
```

Extra space is needed for any via with three or more adjacent cuts that are within 0.1. The adjacent vias require spacing greater than or equal to 0.11.

In the following figure, each of the cases shows at most two adjacent vias to any given via, therefore only default spacing is needed.

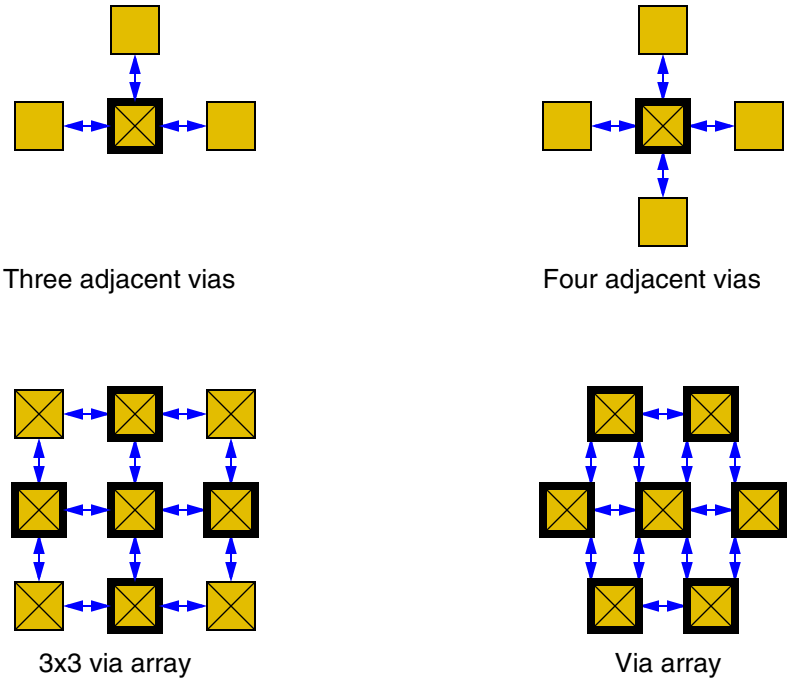
Illustration for minAdjacentViaSpacing Edge-to-edge



Diagonal spacing is greater than the **distance** parameter.

In the following figure, all the cases show one or more cuts with three or more adjacent cuts and require the `minAdjacentViaSpacing` spacing, instead of the smaller default spacing.

Illustration of minAdjacentViaSpacing with numCuts



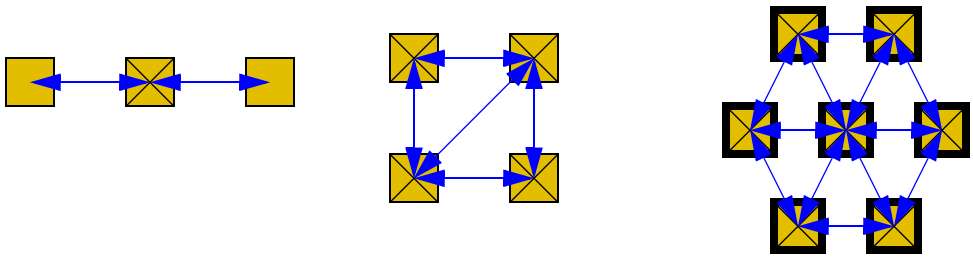
Example 2: minAdjacentViaSpacing Center-to-Center

```
set_constraint_parameter -name numCuts -IntValue 2
set_constraint_parameter -name distance -Value 0.2
set_constraint_parameter -name oaCenterToCenter -BoolValue true
set_layer_constraint -constraint minAdjacentViaSpacing -layer VIA1  -Value 0.2
```

Extra space is needed for any via with two or more adjacent cuts that are within 0.2, measured center-to-center. The adjacent vias require spacing greater than or equal to 0.2, measured center-to-center.

The following figure illustrates how center-to-center measurements are made for determining the number of adjacent vias within `distance` and the spacing needed for vias that meet the criteria for the `minAdjacentViaSpacing` constraint.

Illustration for minAdjacentViaSpacing with oaCenterToCenter

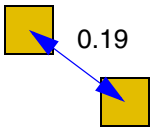


Example 3: minAdjacentViaSpacing with noParallelRunLength

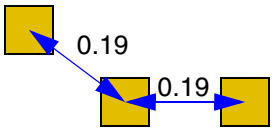
```
set_constraint_parameter -name numCuts -IntValue 2
set_constraint_parameter -name distance -Value 0.2
set_constraint_parameter -name oaCenterToCenter -BoolValue true
set_constraint_parameter -name noParallelRunLength -BoolValue true
set_layer_constraint -constraint minAdjacentViaSpacing \
  -layer VIA1  -Value 0.2
```

Specifies that the minimum adjacent via spacing measured center-to-center must be 0.2 user units if there are at least two adjacent vias with no parallel run length within 0.2 user units.

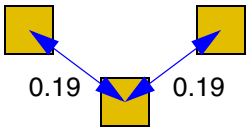
Illustration for minAdjacentViaSpacing with noParallelRunLength



a) PASS. Only one neighbor cut is within 0.2 user units.



b) PASS. The right cut has parallel run length > 0 with the middle cut, and is not counted as a neighbor.



c) FAIL. The middle cut has two neighbors with no parallel run length so the constraint applies and a minimum spacing of 0.2 user units is required, measured center-to-center.

Related Topics

Via Construction Constraints

check_space

minCutClassClearance

Specifies the minimum spacing between cuts of different classes on different layers.

minCutClassClearance Quick Reference

Constraint Type	Layer pair (symmetric)
Value Types	TwoDTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction
Group Operators	AND, OR

Value Type

TwoDTblValue	Specifies the minimum spacing between cuts of different classes on different layers. The column header (`-TblCols`) and the row headers for the 2-D table specify the width and length for each cut class.

Optional Parameters

`cutClassCenterToCenter`

Specifies whether the cut spacing is measured from cut-center-to-cut-center (1) or cut-edge-to-cut-edge (0). By default, cut spacing is performed cut-edge-to-cut-edge. The column header (`-TblCols`) and the row headers for the 2-D table specify the width and length for each cut class.

Type: Int2DTblValue

`defaultCutSpacing`

Specifies the default cut spacing between cut classes.

Type: Value, in user units

`exceptConnectivityType`

Specifies the vias for which this constraint does not apply, based on their connectivity.

Type: StringAsIntValue

`sameNetConnectivity` (same net)	1
`sameIslandConnectivity` (same metal)	2
`directConnectShapesConnectivity` (directly connected by same metal)	3
`sameViaConnectivity` (same metal above and below)	4

Note: This parameter is mutually exclusive with `oaConnectivityType`.

`oaConnectivityType`

Specifies the shapes to which this constraint applies, based on their connectivity. Figure on page 864 shows a graphical example of each connectivity type.

Type: StringAsIntValue

`anyConnectivity` (any)	0
`sameNetConnectivity` (same net)	1
`sameIslandConnectivity` (same metal)	2
`directConnectShapesConnectivity` (directly connected by same metal)	3

`parallelRunLength`

Specifies that the constraint applies only if the parallel run length (prl) between the two via cuts is greater than or equal to this value. Both positive and negative values are allowed.

If multiple `minCutClassClearance` constraints are defined in an AND constraint group, the constraint is applied by evaluating the prl values, specified in the ascending order, to locate the first prl value for which the actual prl is greater than or equal to the parameter value. As a result, only one constraint in an AND constraint group applies. For example, in an AND constraint group with prl values specified as -1.0, -0.05, 0, 0.05, 1, the constraint value that applies is determined as follows:

Actual length (prl)	**Use constraint value for length**
<-1	The constraint does not apply.
>=-1 and <-0.05	-1.0
>=-0.05 and <0	-0.05
>=0 and <0.05	0
>=0.05 and <1.0	0.05
>=1.0	1.0

Type: `Value`, in user units

`nonZeroEnclosure`

Specifies that the minimum cut class spacing between cut classes on different layers optionally applies only to the cut edges of the layer1 shape enclosed by the metal layer above with an enclosure greater than 0, and that the cut edges have a nonzero parallel overlap with cut shapes on layer2. The enclosing metal layer condition does not apply to cut shapes that do not have a parallel overlap.
Type: `LayerValue`

`nonCutClassEdgeSpacing`

Specifies the spacing between cut class shapes and non-cut class shapes (for example, blockages).

Type: `ValueArrayValue`, in user units

`cutClassProfile` Specifies an extension distance that is applied to cut shapes of certain cut classes before cut spacing is measured. The table layout is identical to the constraint value table. This parameter is valid only if there is no parallel overlap between cut shapes.

Type: `Int2DTblValue`, in user units

`cutClassSizeBy` Specifies the cut size required for the cut class. If there are *n* rows and *m* columns in the constraint value table, this parameter should specify (*n*+*m*) extension values.

Type: `ValueArrayValue`, in user units

Note: `cutClassSizeBy` is mutually exclusive with `cutClassProfile`.

`cutClassList` Stores the cut class dimensions in an array.

Type: `ValueArrayValue`, in user units

`cutClassMeasureType` Specifies that the constraint should be applied as specified for each cut class combination.

- `centerToCenter` indicates that the spacing value must be measured center-to-center.
- `centerAndEdge` indicates that the maximum of all applicable values should be measured center-to-center, and the minimum of all applicable values should be measured edge-to-edge.
- `centerAndEdgeNoPRL` indicates that the spacing between two cuts should be the greater of the following two values:
  - The maximum of all applicable values, measured center-to-center.
  - The minimum of all applicable values, measured edge-to-edge.

Type: `Int2DTblValue`

Note: Only one of the values listed above can be used in a table. A combination of values in not allowed.

`exactAlignedSpacing` Specifies an array of DualValue pairs. The first value in each pair specifies the width of a square cut class. The second value specifies the spacing (or clearance) required between completely aligned horizontal or vertical cuts of the cut class.

Type: `DualValueTbl`, in user units

Examples

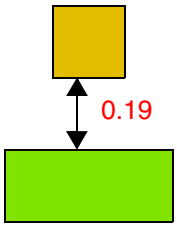
```
# edge-to-edge spacing by default

# parallel run length >= 0
set_constraint_parameter -name parallelRunLength -Value 0
set_layerpair_constraint -constraint minCutClassClearance \
  -layer1 V1 -layer2 V2 \
  -TblCols           {  0.10 0.10 0.10 0.25 } \
  -TwoDTblValue { 0.10  0.00 0.00 0.00 0.20   \
                  0.10  0.00 0.00 0.00 0.20   \
                  0.10  0.00 0.00 0.00 0.00   \
                  0.25  0.20 0.20 0.00 0.00   }
```

Sets the minimum clearance between VA (0.1 x 0.1) and VB (0.1 x 0.25) cut classes on V1 and V2 as follows:

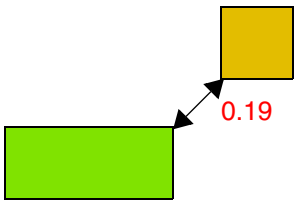
- 0.20 user units is the edge-to-edge spacing between the side (long) edge of a VB via to a VA via.
- 0.00 user units is the edge-to-edge spacing for all other combinations.

Illustration for minCutClassClearance with parallelRunLength

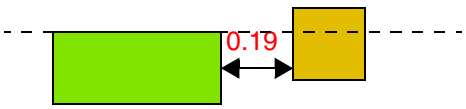
```
minCutClassClearance  V1  V2
   {       0.10 0.10 0.10 0.25   }
   { 0.10  0.00 0.00 0.00 0.20   \
     0.10  0.00 0.00 0.00 0.20   \
     0.10  0.00 0.00 0.00 0.00   \
     0.25  0.20 0.20 0.00 0.00   }
   parallelRunLength 0
```



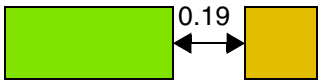
a) FAIL. VA to VB side edge requires 0.20 spacing.



b) FAIL. VA to VB side edge requires 0.20 spacing, with or without parallel edge overlap > 0.



c) FAIL. VA overlaps projection line of VB side edge.



d) PASS. There is no overlap with the projection lines of VB side edges.

Related Topics

Via Construction Constraints

minCutClassSpacing

Specifies the minimum spacing between cuts of different classes on the same layer. This constraint enables cut class aware spacing rules. Spacing requirements depend on the following factors:

- Size of the cut (the cut class).
- For rectangular cuts, which edge (long or short) is being used.
- Whether the two cuts are on the same metal shape (sameMetal) or on the same net (sameNet)
- Whether the rules are being specified as center-to-center or edge-to-edge.
- Whether the parallel run length is greater than 0

In some processes, the spacing between two square cuts of a particular cut class which are perfectly aligned vertically or horizontally is different from the spacing for unaligned cuts.

minCutClassSpacing Quick Reference

Constraint Type	Layer
Value Types	`TwoDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction
Group Operators	AND, OR

Value Type

`TwoDTblValue`	Specifies the minimum spacing between cuts of different classes on the same layer. The column header (`-TblCols`) and the row headers for the 2-D table (`TwoDTblValue`) specify the width and length for each cut class.

Required Parameters

None

Optional Parameters

`cutClassCenterToCenter`

Specifies whether the cut spacing is measured from cut-center-to-cut-center (1) or cut-edge-to-cut-edge (0). By default, cut spacing is performed cut-edge-to-cut-edge. The column header (`-TblCols`) and the row headers for the 2-D table specify the width and length for each cut class.

Type: `Int2DTblValue`

`cutClassList`

Stores the cut class dimensions in an array of DualValue, each with the width and length of a cut class.

Type: `DualValueTbl`

`cutClassMeasureType`

Specifies how the constraint should be applied for each cut class combination as `centerAndEdge` (2) or `centerAndEdgeNoPRL` (3). The table layout is identical to the constraint value table.

- `centerAndEdge` (2): The maximum of all applicable values should be measured center-to-center. The minimum of all applicable values should be measured edge-to-edge.
- `centerAndEdgeNoPRL` (3): The spacing between two cuts should be the greater of the following two values: the maximum of all applicable values, measured center-to-center; and the minimum of all applicable values, measured edge-to-edge.

Type: `TwoDTblValue`

`cutClassProfile`

Specifies an extension distance applied only to cut shapes of rectangular cut classes before measuring cut spacing. The table layout is identical to the constraint value table.

Type: `TwoDTblValue`

`defaultCutSpacing`

Specifies the default cut spacing between cut classes.

Type: `Value`

`exactAlignedSpacing` Specifies an array of DualValue pairs, if set. The first value in each DualValue is the width of a square cut class. The second value is the spacing (or clearance), in user units, required between completely aligned horizontal or vertical cuts of the cut class.

Type: `DualValueTbl`

`exceptConnectivityType`

Determines the vias to which this constraint does not apply. This parameter is mutually exclusive with `oaConnectivityType`.

String	Integer Value
`sameNetConnectivity`	1
Constraint does not apply to same net only	
`sameIslandConnectivity`	2
Constraint does not apply to via cuts connected by contiguous same metal	
`directConnectShapesConnectivity`	3
Constraint does not apply to via cuts directly connected by the same metal	
`sameViaConnectivity`	4
Constraint does not apply to via cuts with the same metal above and same metal below	

Type: `StringAsIntValue`

`oaConnectivityType`

Determines the shapes to which the constraint applies. This parameter is mutually exclusive with `exceptConnectivityType`.

String	Integer Value
`anyConnectivity` (default) Constraint applies to any connectivity	0
`sameNetConnectivity` Constraint applies to same net only	1
`sameIslandConnectivity` Constraint applies to via cuts connected by contiguous same metal	2
`directConnectShapesConnectivity` Constraint applies to via cuts directly connected by the same metal	3
`sameViaConnectivity` Constraint applies to via cuts with the same metal above and same metal below	4

Type: `StringAsIntValue`

`parallelRunLength`

Specifies that the constraint applies only if the parallel run length (prl) between the two via cuts is greater than or equal to this value. Both positive and negative values are allowed.

If multiple `minCutClassSpacing` constraints are defined in an AND constraint group, the constraint is applied by evaluating the prl values, specified in the ascending order, to locate the first prl value for which the actual prl is greater than or equal to the parameter value. As a result, only one constraint in an AND constraint group applies. For example, in an AND constraint group with prl values specified as -1.0, -0.05, 0, 0.05, 1, the constraint value that applies is determined as follows:

Actual length (prl)	**Use constraint value for length**
<-1	The constraint does not apply.
>=-1 and <-0.05	-1.0
>=-0.05 and <0	-0.05
>=0 and <0.05	0
>=0.05 and <1.0	0.05
>=1.0	1.0

Type: `Value`

`oaSpacingDirection`

Specifies the direction in which the completely aligned spacing applies.

Valid Values:

`anyDirection`	0	Horizontally and vertically (default)
`horizontalDirection`	1	Horizontally only
`verticalDirection`	2	Vertically only

The `oaSpacingDirection` parameter can only be specified if the `exactAlignedSpacing` parameter is defined.

Type: `StringAsIntValue`

`sameMask`

Specifies whether the constraint applies only between shapes on the same mask (`true`) or between all shapes on the given layer (`false`, the default).

Type: `BoolValue`

`nonCutClassEdgeSpacing`

Specifies the spacing, in user units, between cutClass shapes and non-cutClass shapes (for example, blockages).

Type: `ValueArrayValue`

`bothNegativeParallelRunLengths`

Allows both horizontal and vertical offsets between the cut shapes to be used as parallel run lengths, if the parallel run length between the cut shapes is less than zero.

When this parameter is specified, two separate checks are performed using the two offset values. If horizontal offset is used as the parallel run length, required spacing is looked up by taking into account the vertical edges facing each other, and when vertical offset is used as the parallel run length, required spacing is looked up by taking into account the horizontal edges facing each other. The maximum of the spacing values thus obtained is the required spacing.

If this parameter is not specified, the smaller of the two offsets is considered as the parallel run length and spacing is looked up using this value. Because the edges that are facing each other cannot be determined, the maximum of the four possible spacing values is compared with the Euclidean spacing.

Note: If this parameter is specified for one `minCutClassSpacing` constraint defined on a particular layer, it must be specified for all `minCutClassSpacing` constraints defined on that layer.

Type: `BoolValue`

Examples

- Example 1: minCutClassSpacing for a Square Cut Class
- Example 2: minCutClassSpacing for a Square and a Rectangular Cut Class
- Example 3: minCutClassSpacing for a Rectangular Cut Class
- Example 4: minCutClassSpacing for Three Cut Classes
- Example 5: minCutClassSpacing with nonCutClassEdgeSpacing
- Example 6: minCutClassSpacing with bothNegativeParallelRunLengths

Example 1: minCutClassSpacing for a Square Cut Class

```
# default is edge-to-edge measurements. Change this to
# center-to-center (value of 1 enables center-to-center)
set_constraint_parameter -name cutClassCenterToCenter \
  -TblCols            {  0.10 0.10 } \
  -Int2DTblValue { 0.10  1    1      \
                   0.10  1    1 }
# parallel run length can be >= 0. This is the default.
set_constraint_parameter -name parallelRunLength -Value 0
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 \
  -TblCols            {  0.10 0.10 } \
  -TwoDTblValue  { 0.10  0.20 0.20   \
                   0.10  0.20 0.20 }
```

Specifies 0.20 user units as the cut-to-cut spacing between two VA vias (0.10x0.10), measured center-to-center.

a) FAIL. The center-to-center spacing between the two shapes is 0.19 (<0.2).

b) PASS. The center-to-center spacing between the two shapes is 0.2

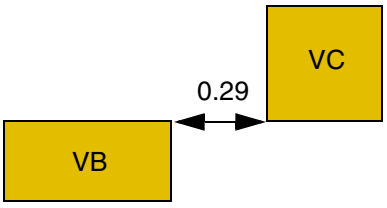
Example 2: minCutClassSpacing for a Square and a Rectangular Cut Class

```
create_constraint_group -name cutClassSpacing -opType or -db tech
add_constraint_group -subGroupName cutClassSpacing -groupType foundry
# edge-to-edge spacing by default
# case 1: no parallel run length
set_constraint_parameter -name parallelRunLength -Value -0.001
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 -group cutClassSpacing \
  -TblCols            {  0.10 0.25 0.25 0.25 } \
  -TwoDTblValue { 0.10  0.15 0.15 0.15 0.15   \
                  0.25  0.15 0.15 0.15 0.15   \
                  0.25  0.15 0.15 0.50 0.50   \
                  0.25  0.15 0.15 0.50 0.50} -create true
# case 2: parallel run length > 0
set_constraint_parameter -name parallelRunLength -Value 0.001
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 -group cutClassSpacing \
  -TblCols            {  0.10 0.25 0.25 0.25 } \
  -TwoDTblValue { 0.10  0.40 0.40 0.30 0.30   \
                  0.25  0.40 0.15 0.15 0.15   \
                  0.25  0.30 0.15 0.50 0.50   \
                  0.25  0.30 0.15 0.50 0.50} -create true
```

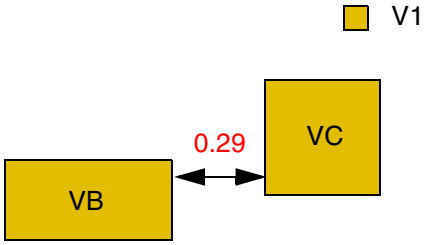
Assumes that a rectangular cut class (0.1x0.25) and square cut class (0.25x0.25) are defined. The following conditions apply:

- 0.30 user units is the edge-to-edge spacing between the end edge (0.10) of a VB via and a VC via with parallel edge overlap > 0
- 0.40 user units is the edge-to-edge spacing between the end edge of a VB via to another VB via with parallel edge overlap > 0
- 0.50 user units is the edge-to-edge spacing between a VC via to another VC via
- 0.15 user units is the edge-to-edge spacing for the rest of the combinations

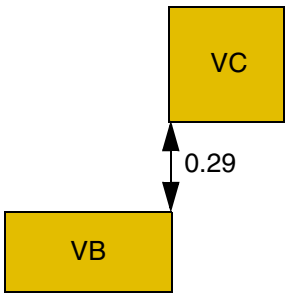
To specify the constraint using Tcl, two `minCutClassSpacing` constraints must be set in an OR constraint group, one for cuts with no parallel run length (`parallelRunLength` parameter set to -0.001) and the other for cuts with parallel run length > 0 (`parallelRunLength` parameter set to 0.001). When setting the constraint values, include `-create true` to ensure that each constraint value is added, instead of overwriting a previously set value.



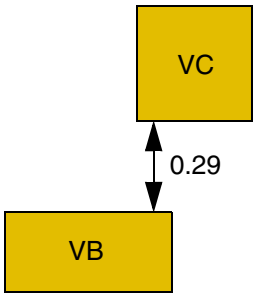
a) PASS. The parallel run length between a VB end edge and a VC edge is zero. Therefore, a default minimum spacing of 0.15 applies (0.29 > 0.15).



b) FAIL. The spacing between a VB end edge and a VC edge must be at least 0.30 (0.29 < 0.30).



c) PASS. The parallel run length between a VB end edge and a VC edge is zero. Therefore, a default minimum spacing of 0.15 applies (0.29 > 0.15).



d) PASS. A default minimum spacing of 0.15 (0.29 > 0.15) applies between a VB side edge and a VC edge.

Example 3: minCutClassSpacing for a Rectangular Cut Class

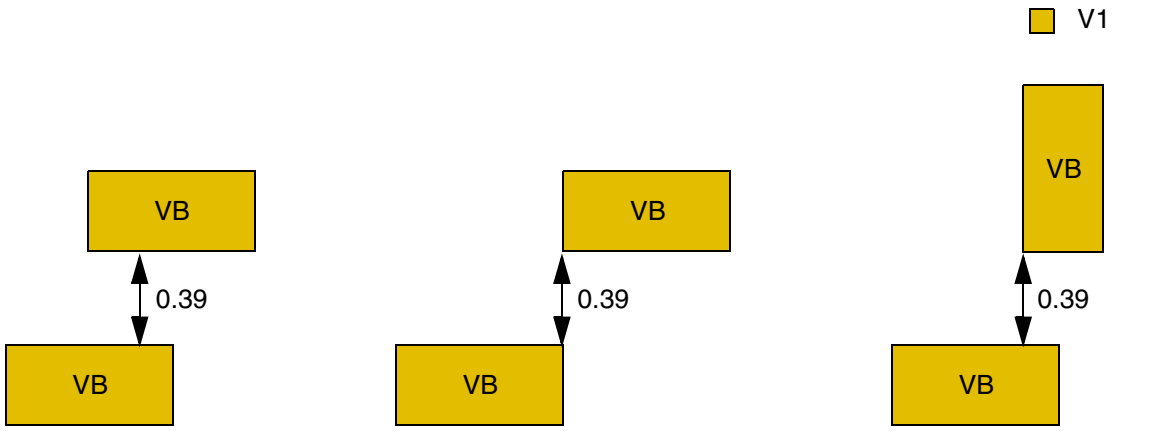
```
create_constraint_group -name cutClassSpacing -opType or -db tech
add_constraint_group -subGroupName cutClassSpacing -groupType foundry
# edge-to-edge spacing by default
# case 1: no parallel run length
set_constraint_parameter -name parallelRunLength -Value -0.001
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 -group cutClassSpacing \
  -TblCols           {  0.10 0.25 } \
  -TwoDTblValue { 0.10  0.15 0.15  \
                  0.25  0.15 0.15  } -create true
# case 2: parallel run length > 0
set_constraint_parameter -name parallelRunLength -Value 0.001
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 -group cutClassSpacing \
  -TblCols           {  0.10 0.25 } \
  -TwoDTblValue { 0.10  0.15 0.15  \
                  0.25  0.15 0.40  } -create true
```

Assumes that a rectangular cut class (VB) is defined (0.1x0.25). The following conditions apply:

- 0.40 user units is the edge-to-edge spacing between the side edge of a VB via to a side edge of another VB via with parallel edge overlap > 0
- 0.15 user units is the edge-to-edge spacing for all other VB of the combinations

To specify the constraint using Tcl, two `minCutClassSpacing` constraints must be set in an OR constraint group, one for cuts with no parallel run length (`parallelRunLength` parameter set to -0.001) and the other for cuts with parallel run length > 0 (`parallelRunLength` parameter set to 0.001). When setting the constraint values, include

`-create true` to ensure that each constraint value is added, instead of overwriting a previously set value.



a) FAIL. The spacing between two VB side edges must be at least 0.40 (0.39 < 0.40).

b) PASS. The spacing required between a VB side edge and a VB end edge is 0.15 (0.39 > 0.15).

c) PASS. The spacing required between a VB side edge and a VB end edge is 0.15 (0.39 > 0.15).

Example 4: minCutClassSpacing for Three Cut Classes

```
create_constraint_group -name cutClassSpacing -opType or -db tech
add_constraint_group -subGroupName cutClassSpacing -groupType foundry

# This section sets the constraint with parallel run length = 0 (no overlap)
set_constraint_parameter -name parallelRunLength -Value -0.001
set_constraint_parameter -name cutClassCenterToCenter \
  -TblCols            {  0.1  0.1  0.1  0.25 0.25 0.25} \
  -Int2DTblValue { 0.1  1    1    0    0    0    0
                   0.1  1    1    0    0    0    0
                   0.1  0    0    0    0    0    0
                   0.25 0    0    0    0    0    0
                   0.25 0    0    0    0    1    1
                   0.25 0    0    0    0    1    1 }
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 -hardness hard -group cutClassSpacing -create true \
  -TblCols            {  0.1  0.1  0.1  0.25 0.25 0.25} \
  -TwoDTblValue  { 0.1  0.2  0.2  0.15 0.15 0.15 0.15
                   0.1  0.2  0.2  0.15 0.15 0.15 0.15
                   0.1  0.15 0.15 0.15 0.15 0.15 0.15
                   0.25 0.15 0.15 0.15 0.15 0.15 0.15
                   0.25 0.15 0.15 0.15 0.15 0.5  0.5
                   0.25 0.15 0.15 0.15 0.15 0.5  0.5 }
# This section sets the constraint with parallel run length > 0
set_constraint_parameter -name parallelRunLength -Value 0.001
set_constraint_parameter -name cutClassCenterToCenter \
  -TblCols            {  0.1  0.1  0.1  0.25 0.25 0.25} \
  -Int2DTblValue { 0.1  1    1    0    0    0    0
                   0.1  1    1    0    0    0    0
                   0.1  0    0    0    0    0    0
```

```
                    0.25 0    0    0    0    0    0
                    0.25 0    0    0    0    1    1
                    0.25 0    0    0    0    1    1 }
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 -hardness hard -group cutClassSpacing -create true \
  -TblCols            {  0.1  0.1  0.1  0.25 0.25 0.25} \
  -TwoDTblValue  { 0.1  0.2  0.2  0.3  0.15 0.15 0.15
                   0.1  0.2  0.2  0.3  0.15 0.15 0.15
                   0.1  0.3  0.3  0.4  0.4  0.3  0.3
                   0.25 0.15 0.15 0.4  0.15 0.15 0.15
                   0.25 0.15 0.15 0.3  0.15 0.5  0.5
                   0.25 0.15 0.15 0.3  0.15 0.5  0.5 }
```

Assumes three cut classes are defined for the layer: VA, VB, and VC. VA and VC are square cut classes (0.1x0.1 and 0.25x0.25, respectively). VB is a rectangular cut class (0.1x0.25). The following conditions apply:

- 0.20 user units is the cut-to-cut spacing between 2 VA vias (0.1x0.1), measured center-to-center
- 0.50 user units is the cut-to-cut spacing between 2 VC vias (0.25x0.25), measured center-to-center
- 0.30 user units is the edge-to-edge spacing between the end edge (0.1) of a VB via and a VA or VC via with parallel edge overlap > 0
- 0.40 user units is the edge to edge spacing between the end edge (0.1) of a VB via & any edges of another VB via with parallel edge overlap > 0
- 0.15 user units is the edge-to-edge spacing for the rest of the combinations. Notice that it is a symmetrical table. For instance, both the VA column to VB END and the VB END column to VA have 0.30 μm when parallel edge overlap > 0

To specify the constraint using Tcl, two `minCutClassSpacing` constraints must be set in an OR constraint group, one for cuts with no parallel run length (`parallelRunLength` parameter set to -0.001) and the other for cuts with parallel run length > 0 (`parallelRunLength` parameter set to 0.001). For each constraint, parameters must be set to specify center-to-center or edge-to-edge cut spacing and the parallel run length, which applies to the cut spacing. When setting the constraint value, include `-create true` to ensure that each constraint value is added, instead of overwriting a previously set value.

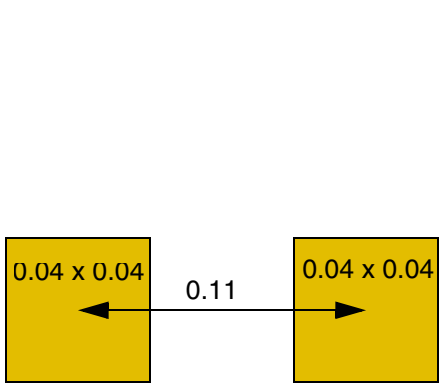
Example 5: minCutClassSpacing with nonCutClassEdgeSpacing

```
# nonCutClassEdgeSpacing
# edge-to-edge spacing for a 0.04x0.04 cut class or
# a 0.04x0.08 cut class to a non-cut class shape (cut or blockage)
# pairs are <edgelength> <spacing>
# first two pairs  are for 0.04x0.04 square cut class; spacing 0.2
# second two pairs are for 0.04x0.08 rectangular cut class
#   the spacing to the short edge (0.04) is 0.22
#   the spacing to the  long edge (0.08) is 0.25
```

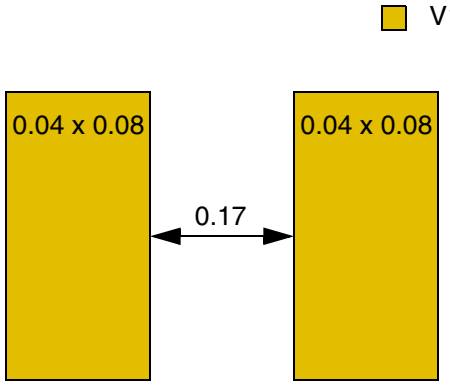
```
set_constraint_parameter -name nonCutClassEdgeSpacing \
 -valueArrayValue { 0.04 0.2  \
                    0.04 0.2  \
                    0.04 0.22 \
                    0.08 0.25 }
# default is edge-to-edge measurements. Change this to
# center-to-center (value of 1 enables center-to-center)
set_constraint_parameter -name cutClassCenterToCenter \
  -TblCols             {  0.04 0.04 0.04 0.08} \
  -Int2DTblValue { 0.04  1    1    0    0     \
                   0.04  1    1    0    0     \
                   0.04  0    0    0    0     \
                   0.08  0    0    0    0     }
# constraint value
set_layer_constraint -constraint minCutClassSpacing \
  -layer V1 \
  -TblCols             {  0.04  0.04  0.04  0.08} \
  -TwoDTblValue  { 0.04  0.1   0.1   0.15  0.15  \
                   0.04  0.1   0.1   0.15  0.15  \
                   0.04  0.15  0.15  0.18  0.18  \
                   0.08  0.15  0.15  0.18  0.18  }
```

Assumes two cut classes are defined for the layer: VA and VB. VA is a square cut class (0.04x0.04). VB is a rectangular cut class (0.04x0.08). The following conditions apply:

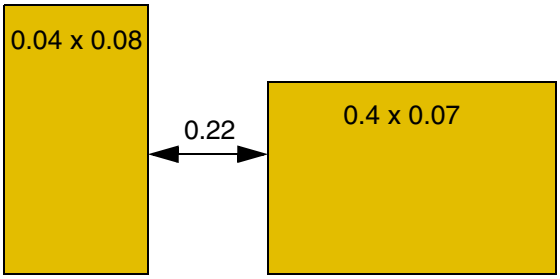
- 0.1 user units is the cut-to-cut spacing between two VA vias (0.04x0.04), measured center-to-center
- 0.15 user units is the edge-to-edge spacing between the end edge (0.04) or side edge (0.08) of a VB via and a VA via
- 0.18 user units is the edge-to-edge spacing between two VB vias
- 0.2 user units is the spacing between a VA via and a non-cut class shape
- 0.22 user units is the spacing between the end edge (0.04) of a VB via and a non-cut class shape
- 0.25 user units is the spacing between the side edge (0.08) of a VB via and a non-cut class shape



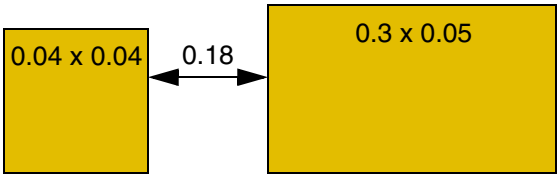
a) PASS. The required 0.04x0.04 center-to-center spacing is 0.1 and is met (0.11 > 0.1).



b) FAIL. The required 0.04x0.08 long side edge-to-edge spacing is 0.18 and is not met (0.17 < 0.18).



c) FAIL. The right shape is not a cutClass shape so the `nonCutClassEdgeSpacing` applies. The 0.04x0.08 cut requires an edge-to-edge spacing of 0.25 on the long side which is not met.



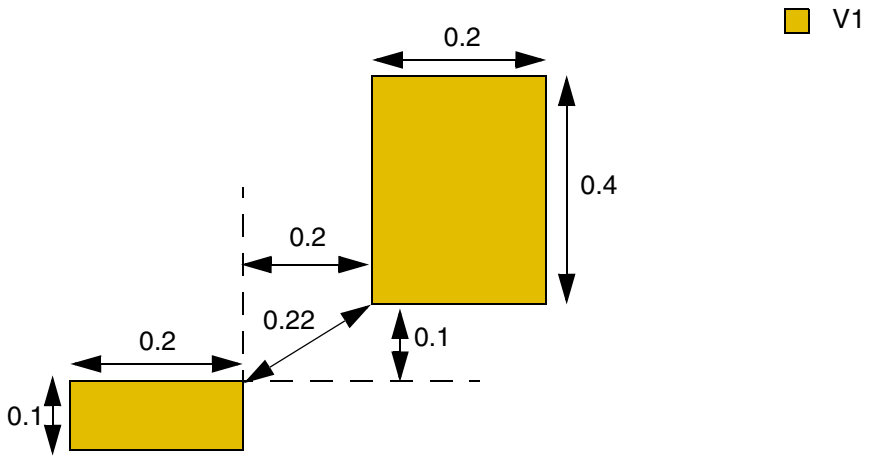
d) FAIL. The right shape is not a cutClass shape so the `nonCutClassEdgeSpacing` applies. The 0.04x0.04 cut requires an edge-to-edge spacing of 0.2 on the long side which is not met.

Example 6: minCutClassSpacing with bothNegativeParallelRunLengths

```
create_constraint_group -name cutClassSpacing -opType or -db tech
add_constraint_group -subGroupName cutClassSpacing -groupType foundry
# This section sets the constraint with parallel run length = -0.1
set_constraint_parameter -name parallelRunLength -Value -0.1
set_constraint_parameter -name bothNegativeParallelRunLengths -BoolValue true
set_layer_constraint -constraint minCutClassSpacing \
   -layer V1 -hardness hard -group cutClassSpacing -create true \
   -TblCols             {  0.1   0.2   0.2   0.4} \
   -TwoDTblValue  { 0.1   0.2   0.2   0.2   0.22 \
                    0.2   0.2   0.2   0.21  0.23 \
                    0.2   0.2   0.21  0.3   0.3  \
                    0.4   0.21  0.23  0.3   0.3  }
# This section sets the constraint with parallel run length = -0.2
set_constraint_parameter -name parallelRunLength -Value -0.2
set_constraint_parameter -name bothNegativeParallelRunLengths -BoolValue true
set_layer_constraint -constraint minCutClassSpacing \
   -layer V1 -hardness hard -group cutClassSpacing -create true \
   -TblCols             {  0.1  0.2   0.2   0.4} \
   -TwoDTblValue  { 0.1   0.2  0.2   0.2   0.22 \
```

```
0.2   0.2  0.2   0.21  0.24 \
0.2   0.2  0.21  0.3   0.3  \
0.4   0.21 0.23  0.3   0.3  }
```

Defines spacing between two cut shapes of type VA (0.2x0.1) and VB (0.2x0.4) with parallel run length (PRL) -0.1 and -0.2.



PASS. Because `bothNegativeParallelRunLengths` is specified, both tables are looked up to determine the spacing required between the two shapes (indicated in blue in the table above).
Horizontal PRL = -0.2 => spacing = 0.21
Vertical PRL = -0.1 => spacing = 0.22
Required spacing is the larger of the two spacing values, that is 0.22. The Euclidean spacing between the shapes is 0.22 (=0.22).

Related Topics

Via Construction Constraints

check_space

minLargeViaArrayCutSpacing

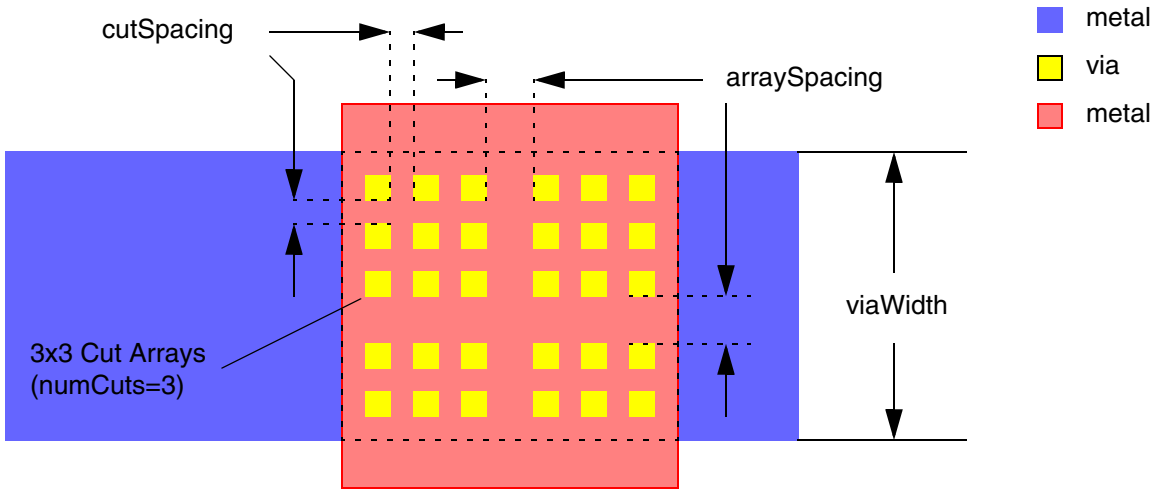
Specifies the minimum distance, edge-to-nearest-edge between cuts in a large via array. This constraint is associated with a cut layer and only applies within via arrays that satisfy the specified minimum and maximum requirement for via cuts.

minLargeViaArrayCutSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`minLargeViaArrayCutSpacing` constraint has a `Value` that represents the cut-to-cut spacing, in user units, within large via arrays that satisfy the specified minimum and maximum requirement for via cuts.



Large Via With Cut Arrays

Format	Example
Tcl	`set_constraint_parameter -name minNumCuts -IntValue` *`i_numCuts`* `set_layer_constraint -constraint minLargeViaArrayCutSpacing \` `-layer` *`s_layer`* `-Value` *`f_spacing`*

Parameters

- `maxNumCuts` (`IntValue`) specifies that the constraint applies only if the number of rows and columns in the via cut array is less than or equal to this value.
- `minNumCuts` (`IntValue`) specifies that the constraint applies only if the number of rows and columns in the via cut array is greater than or equal to this value.

Related Topics

Via Construction Constraints

minLargeViaArraySpacing

Specifies the minimum spacing required between cut arrays (arraySpacing). The constraint only applies to the spacing between cut arrays, not between cuts within an array (cutSpacing). This constraint is associated with a cut layer.

Multiple via array spacing values can be specified for a cut layer, based on different array sizes.

minLargeViaArraySpacing Quick Reference

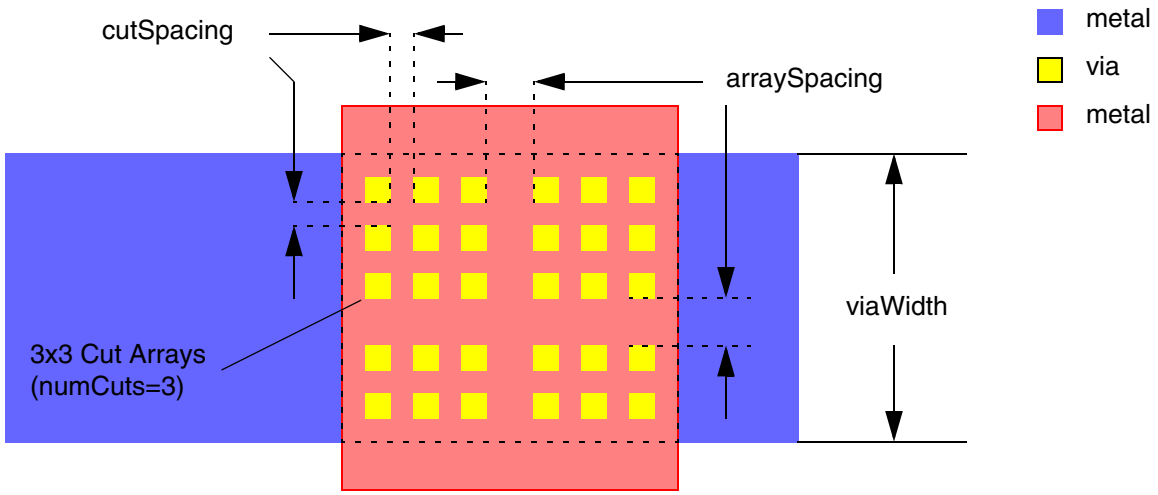
Constraint Type	Layer
Value Types	`OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`minLargeViaArraySpacing` constraints have a `OneDTblValue` to represent the large via array spacing that is required for various cut array sizes. The table lookup key is the number of cuts in the shortest direction of the via array (numCuts in the diagram below) and the table value is the required minimum array spacing.

For example, the number of cuts in a square NxN array is N, rather than the total number of cuts in the array. If neighboring cut arrays have different sizes, the number of cuts in the wider array is used to determine the array-to-array spacing. A cut array can be truncated if it does

not fit within a wire. In this event, the large via array-to-array spacing is still used in placing the truncated array



Large Via With Cut Arrays

Format	Example
Tcl	`set_layer_constraint -constraint minLargeViaArraySpacing \ -layer s_layer -OneDTblValue {{i_numCuts f_spacing}…}`
LEF 5.7	`ARRAYSPACING CUTSPACING cutSpacing {ARRAYCUTS i_numCuts SPACING f_spacing} ... ;`

Parameters

- `cutClass` (`DualValue`, optional) The constraint applies only to the cut class of the given dimensions.
- `parallelOverlap` (`BoolValue`, optional) The constraint applies only when there is a parallel edge overlap > 0.
- `intraArrayCutSpacing` (`IntValue`, optional) If this parameter is specified then the cuts which are spaced by less than this value, are considered to be part of the same array.

Examples

- Example 1 — minLargeViaArraySpacing example

Example 1 — minLargeViaArraySpacing example

This example sets the minimum large via array spacing on `Via1` according to the values in the following table.

Number of Cuts	Minimum Spacing Between Cut Arrays
2	0.10
3	0.20

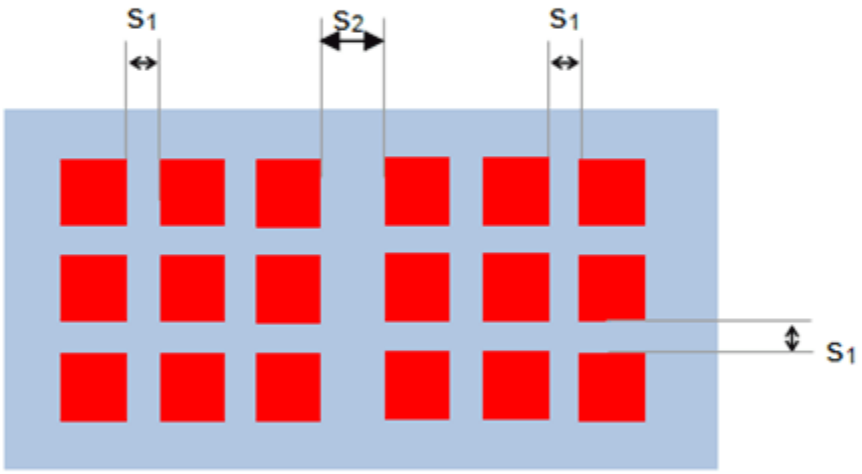
Format	Example
Tcl	`set_layer_constraint -constraint minLargeViaArraySpacing \` `-layer Via1 -OneDTblValue { 2 0.10 3 0.20 }`

Example 2 — minLargeViaArraySpacing example using intraArrayCutSpacing

intraArrayCutSpacing = SC,

Constraint Value = { 1 v1 2 v2 3 v3 4 v4 }

Case 1a.



If s1 <= SC,

S2 >= v3

Related Topics

Via Construction Constraints

minLargeViaArrayWidth

Specifies the minimum width of a via array for it to be considered a large via array and must be built with the spacing given by the minLargeViaArraySpacing constraint. The minimum width applies to the intersection of the top and bottom metal layers in the via that are connected by the cut layer.

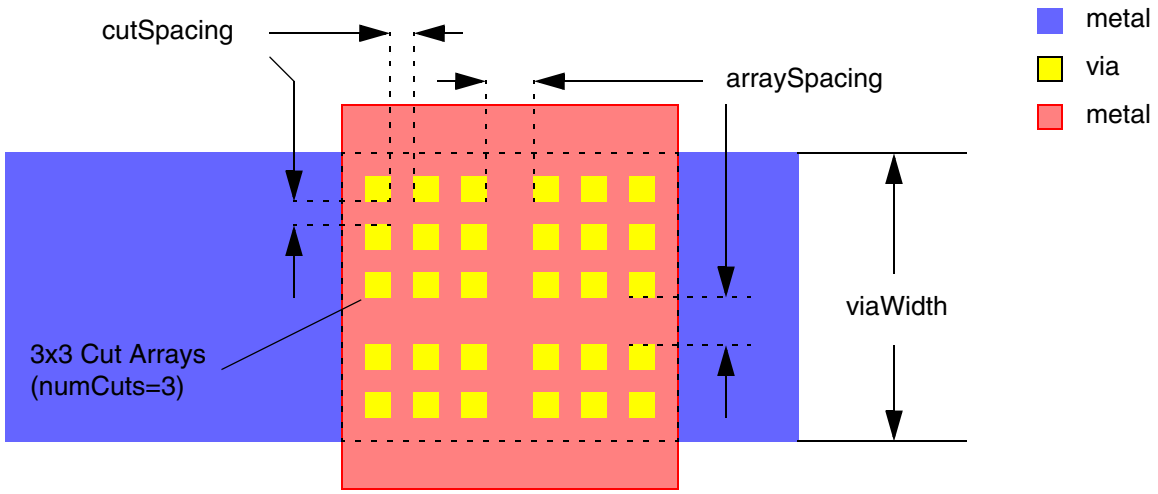
minLargeViaArrayWidth Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

The `minLargeViaArrayWidth` constraint has a `Value` that is the minimum width, in user units, of a large via array.

Format	Example
Tcl	`set_layer_constraint -constraint minLargeViaArrayWidth \ -layer cutLayer -Value viaWidth}`
LEF 5.7	`ARRAYSPACING WIDTH viaWidth`
Virtuoso	`spacings( ( minLargeViaArrayWidth cutLayer viaWidth ) )`



Large Via With Cut Arrays

Parameters

- `cutClass` (`DualValue`, optional) The constraint applies only to the cut class of the given dimensions.
- `parallelOverlap` (`BoolValue`, optional) The constraint applies only when there is a parallel edge overlap > 0.

Examples

Format	Example
Tcl	`set_layer_constraint -constraint minLargeViaArrayWidth \ -layer Via1 -Value 1.2`
LEF 5.7	`ARRAYSPACING WIDTH 1.2`
Virtuoso	`spacings( ( minLargeViaArrayWidth Via1 1.2 ) )`

Related Topics

Via Construction Constraints

minNeighborViaSpacing

Specifies the minimum spacing between two shapes on a given cut layer that have a common neighboring cut shape within a given distance. The distance and spacing is optionally measured from center-to-center of the cut shapes. By default, the distance and spacing are measured from nearest cut edge-to-cut edge.

minNeighborViaSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`minNeighborViaSpacing` constraints have a `Value` that represents the minimum spacing between the two shapes on the cut layer.

Parameters

- `distanceWithin` (`Value`, required) specifies that the constraint applies only when a common neighbor via is less than or equal to this distance from the two cut shapes.
- `oaCenterToCenter` (`BoolValue`, optional) determines whether the spacing and distance are measured center-to-center (`true`) or edge-to-edge (`false`, default).

Examples

This example requires a 0.4 neighbor via spacing if a common neighbor via is within 0.3 user units measured center-to-center.

a) OK. Vias A and B have a common via C within 0.3 and spacing between A and B meets the `minNeighborViaSpacing` of 0.4.
b) Spacing between vias A and C is not subject to `minNeighborViaSpacing` because the distance from via A to via B is greater than the 0.3 `distanceWithin` requirement.
c) Spacing between vias B and C is not subject to `minNeighborViaSpacing` because the distance from via B to via A is greater than the 0.3 `distanceWithin` requirement.

Format	Example

Tcl	`set_constraint_parameter -name centerToCenter -BoolValue true`
	`set_constraint_parameter -name distanceWithin -Value 0.3`
	`set_layer_constraint -constraint minNeighborViaSpacing \`
	`-layer V1 -Value 0.4`

Virtuoso	`spacings(`
	`( minNeighborViaSpacing V1 'within 0.3 'centerToCenter 0.4 )`
	`) ;spacings`

Related Topics

Via Construction Constraints

minParallelViaSpacing

Specifies the minimum spacing for vias that have parallel edges with an overlap greater than 0. This constraint applies to cut layers only when vias do not share the same metal above or below and are on different nets.

Either `minParallelWithinViaSpacing` or `minParallelViaSpacing` can be specified for a cut layer, but not both.

minParallelViaSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`minParallelViaSpacing` constraints have a `Value` that represents the minimum required spacing for vias with overlapping parallel edges on the given cut layer.

Format	Example
Tcl	`set_layer_constraint -constraint minParallelViaSpacing \` ` -layer s_layer -Value f_space`

LEF 5.7	`LAYER s_layer`
	`  TYPE CUT ;`
	`  SPACING f_space PARALLELOVERLAP ;`

Virtuoso	`spacings(`
	`  ( minParallelViaSpacing s_layer f_space )`
	`  )`

Parameters

- `oaConnectivityType` (`StringAsIntValue`, optional) specifies the shapes to which this constraint applies, based on their connectivity, as given in oaConnectivityType Parameter Values for Via Spacing Constraints. This parameter is mutually exclusive with `exceptConnectivityType`.

- `exceptConnectivityType` (`StringAsIntValue`, optional) specifies the vias for which this constraint does not apply, based on their connectivity, as given in exceptConnectivityType Parameter Values. If this argument is not specified, same island vias are excluded. This parameter is mutually exclusive with `oaConnectivityType`.

exceptConnectivityType Parameter Values

String	Integer Value	Constraint does not apply to:
`sameNetConnectivity`	1	Same net only
`sameIslandConnectivity`	2	Via cuts connected by contiguous same metal
`directConnectShapesConnectivity`	3	Via cuts directly connected by the same metal
`sameViaConnectivity`	4	Via cuts with the same metal above and same metal below

Examples

This example sets the minimum spacing to `2.0` between cuts on layer `cut1` that meet the following conditions:

- The cuts are not on the same metal shape

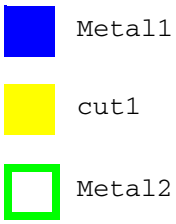
- The cuts have a parallel overlap (adjacent cut edges running parallel to each other have at least some portion of their edges on the same coordinates on the `x` or `y` axis)

Format	Example
Tcl	`set_layer_constraint -constraint minParallelViaSpacing \` ` -layer cut1 -Value 2.0`

LEF 5.7	`LAYER cut1`
	`  TYPE CUT ;`
	`  SPACING 2.0 PARALLELOVERLAP ;`

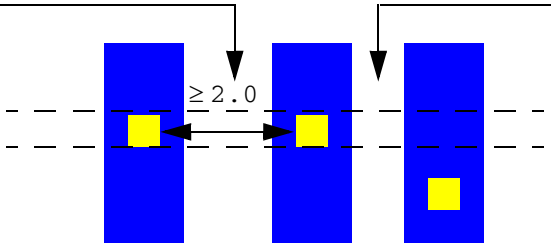
Virtuoso	`spacings(`
	`  ( minParallelViaSpacing cut1 2.0 )`
	`  )`

Assume that `minParallelViaSpacing` is set to `2.0` on the cut1 layer

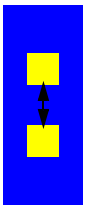


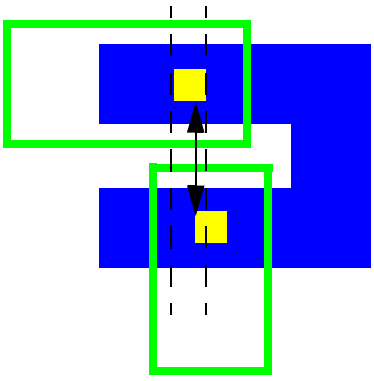
The minimum spacing applies because the cuts on the separate metal shapes have a parallel overlap.

The minimum spacing does not apply because the cuts on the separate metal shapes do not have a parallel overlap.



Although the cuts have a parallel overlap, the minimum spacing does not apply because the cuts are on the same metal shape.





When cuts are on different metal shapes, adjacent cut edges running parallel to each other have at least some portion of their edges on the same coordinates on the x or y axis, the spacing between the cuts must be greater than or equal to the value specified by this constraint.

Related Topics

Via Construction Constraints

minParallelWithinViaSpacing

Specifies the spacing required between a cut and a neighbor cut when the edge of the neighbor cut is *within* a given distance.

Either `minParallelWithinViaSpacing` or `minParallelViaSpacing` can be specified for a cut layer, but not both.

minParallelWithinViaSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

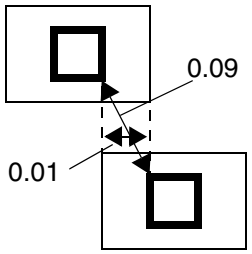
`minParallelWithinViaSpacing` constraints have a `Value` that represents the required spacing between a cut and a neighbor cut when the conditions are met.

Parameters

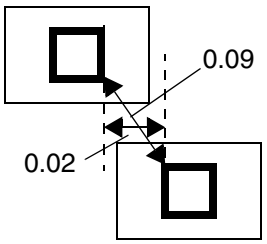
- `oaCutDistance` (`Value`) specifies that the constraint applies only when there is a neighbor cut edge that is less than this distance from the edge of a cut.
- `exceptConnectivityType` (`StringAsIntValue`, optional) specifies that the constraint applies to all vias **except** same net vias. The only valid value for this parameter is `sameNetConnectivity`. By default, the constraint applies to all vias.

Examples

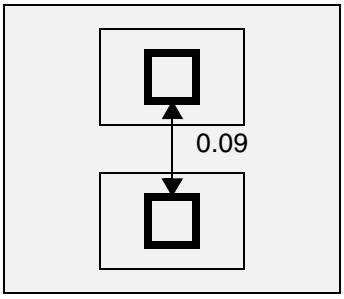
minParallelWithinViaSpacing Example



a) Violation. A neighbor cut edge is within 0.02 of the cut edge, so spacing between cuts must be >= 0.1.



b) OK. The neighbor cut edge is >= 0.02 from the cut edge, so the constraint does not apply.



c) Violation. Neighbor cut edge is < 0.02 from the cut edge; spacing between cuts must be >= 0.1. If exceptConnectivityType = SameNetConnectiivty, there would be no violation in this case.

Format	Example

Tcl	`set_layer_constraint -constraint oaMinViaSpacing \`
	`  -layer V2 -hardness hard -Value 0.09`
	`set_constraint_parameter -name oaCutDistance -Value 0.02`
	`set_layer_constraint -constraint minParallelWithinViaSpacing \`
	`  -layer V2 -hardness hard -Value 0.1`

LEF	`PROPERTY LEF58_SPACING`
	`"SPACING 0.09 ;`
	` SPACING 0.1 PARALLELWITHIN 0.02 ;" ;`

Related Topics

Via Construction Constraints

minRedundantViaSetback

Specifies the metal enclosure required on a metal layer when redundant vias are aligned on the same edge of that metal layer.

minRedundantViaSetback Quick Reference

Constraint Type	Layer pair
Value Types	`OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

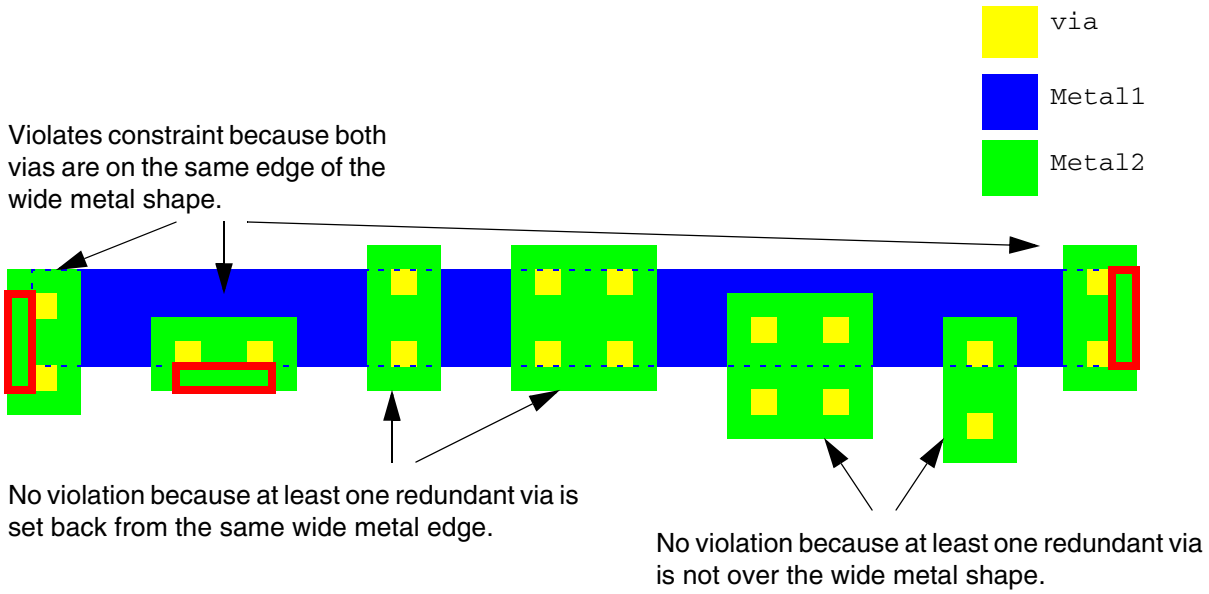
Value Type

`minRedundantViaSetback` constraints have a `OneDTblValue`. The lookup key (`width`) represents the width of the wide metal and the value for the table represents the redundant via wide enclosure width in user units. Layer1 is the cut layer and layer2 is the metal layer.

Format	Example

Tcl	`set_layerpair_constraint -constraint minRedundantViaSetback \`
	` -layer1 s_cutLayer -layer2 s_metalLayer -row_name width \`
	` -OneDTblValue {{f_width f_enclosure}…}`

Virtuoso	`spacingTables(`
	`  ( redundantViaSetback tx_layer1 tx_layer2`
	`  ( ( "width" nil nil ) )`
	`  ( l_table )`
	`  )`
	`  )`



Parameters

- `distanceMeasureType` (`IntValue`, optional) specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1). For information on this parameter, see Euclidean and Manhattan Spacing Constraints.

Related Topics

Via Construction Constraints

minSameMetalSharedEdgeViaSpacing

Specifies the spacing between two via cuts that have a common projection between them and have neighbor wire(s) within *parWithin* distance from them on the same edge. This spacing must be greater than or equal to `oaMinViaSpacing`.

minSameMetalSharedEdgeViaSpacing Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

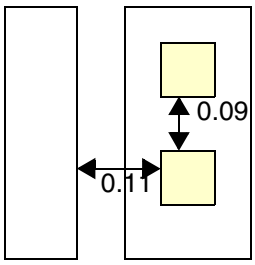
`minSameMetalSharedEdgeViaSpacing` constraints have a `Value` that represents the required spacing.

Parameters

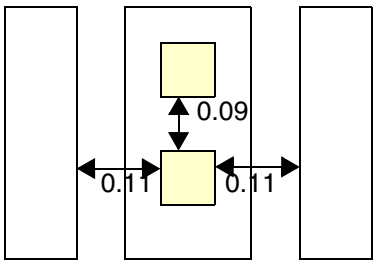
- `parallelEdgeWithin` (`Value`) specifies the *parWithin* distance.
- `aboveOnly` (`BoolValue`, optional) The constraint applies only if the via cuts have a common above metal routing layer.
- `cutClass` (`DualValue`, optional) The constraint applies only to the cut class of the given dimensions.
- `exceptSameViaCount` (`IntValue`, optional) The constraint does not apply if there are greater than or equal to this number of cuts having common same metal on both the above and below layers.
- `exceptTwoEdges` (`BoolValue`, optional) The constraint does not apply if the cuts have two neighbor wires within *parWithin* distance on opposite sides.

Examples

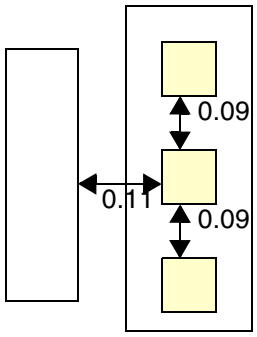
minSameMetalSharedEdgeViaSpacing Examples



a) Violation. There is a neighbor wire within 0.12. The 0.09 spacing between the cuts < the required 0.1 spacing.



b) OK. At least one cut has two neighbors within 0.12 on opposite sides, so the constraint does not apply.



b) OK. There is a neighbor wire within 0.12, but there are three cuts on the same metal above and below so the constraint does not apply.

Format	Example

Tcl	`set_constraint_parameter -name parallelEdgeWithin -Value 0.12`
	`set_constraint_parameter -name exceptTwoEdges -BoolValue true`
	`set_constraint_parameter -name exceptSameVia -IntValue 3`
	`set_layer_constraint -constraint minSameMetalSharedEdgeViaSpacing \`
	`  -layer V2 -hardness hard -Value 0.1`

LEF	`PROPERTY LEF58_SPACING`
	`" SPACING 0.1 SAMEMETALSHAREDEDGE 0.12 EXCEPTTWOEDGES`
	`  EXCEPTSAMEVIA 3;" ;`

Related Topics

Via Construction Constraints

minViaExtension

Specifies the minimum extension of a shape on layer1 over a shape on layer2 for the following cases:

- The layer1 extension on three sides must be greater than or equal to a given value and the extension on the remaining side must be greater than or equal to a second value.
- The extension of two opposite sides must be greater than or equal to a given value, and for the remaining two sides, the extension of one side must be greater than or equal to a second value and the extension of the other side must be greater than or equal to a third value.
- A different minimum extension is required for a diagonal edge compared with non-diagonal edges.

minViaExtension Quick Reference

Constraint Type	Layer pair (non-symmetric)
Value Types	`ValueArrayValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`minViaExtension` constraints have a `ValueArrayValue` that represents the minimum extensions required for the `viaExtensionType` parameter.

minViaExtension ValueArrayValue by viaExtensionType

viaExtensionType	ValueArrayValue
`oneThree`	{`Value` `Value`} where the first value is the minimum extension for three sides and the second value is the minimum extension for the fourth side.
`oneOneTwo`	{`Value` `Value` `Value`} where the first value is the minimum extension for two opposite sides, and the second and third values are the minimum extensions for the other two sides.

viaExtensionType	ValueArrayValue
`diagonal`	{`Value` `Value`} where the first value is the minimum extension for non-diagonal edges and the second is the minimum extension for diagonal edges. {`Value` `Value` `Value`} where the first value is the minimum extension for three sides, the second value is the minimum extension for the fourth side, and the third value is the minimum extension for diagonal edges. {`Value` `Value` `Value` `Value`} where the first value is the minimum extension for two opposite sides, the second and third values are the minimum extensions for the other two non-diagonal sides, and fourth value applies to any diagonal edges.

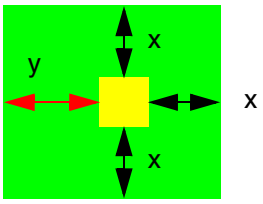
Parameters

- `viaExtensionType` (`StringAsIntValue`, required) specifies the type of minimum via extension that the constraint represents, as given in Table on page 912.

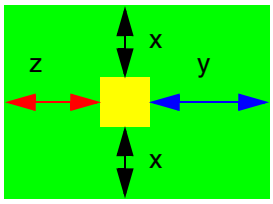
minViaExtension Types

String	Value	Description
`oneThree`	0	Layer1 extension on three sides must be >= x and the fourth side must be y.
`oneOneTwo`	1	Extension on two opposite sides must be >= x, a third side must be >= y and the fourth side must be >= z.
`diagonal`	2	The layer1 extension to the diagonal edge must be >= w and to the non-diagonal edges >= x.

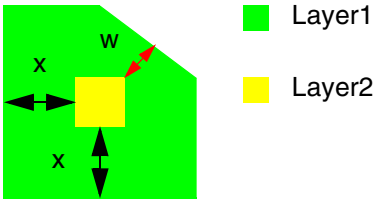
Examples of minViaExtension Types



oneThree: Layer1 extension on three sides must be >= x and the fourth side must be >= y.



oneOneTwo: Extension on two opposite sides must be >= x, a third side must be >= y and the fourth side must be >= z.



diagonal: The layer1 extension to the diagonal edge must be >= w and to the non-diagonal edges >= x.

Examples

The following example sets the minimum via extension for `Metal3` over `Via2`. Three sides must have a minimum extension of 0.04 and the fourth side has a minimum extension of 0.06.

```
set_constraint_parameter -name viaExtensionType -StringAsIntValue oneThree
set_layerpair_constraint -group foundry -layer1 Metal3 -layer2 Via2 \
-constraint minViaExtension -hardness hard -ValueArrayValue { 0.04 0.06}
```

The following example sets the minimum via extension for `Metal3` over `Via3`. Two opposite sides must have a minimum extension of 0.04, and the two remaining sides must have a minimum extension of 0.06 and 0.08.

```
set_constraint_parameter -name viaExtensionType -StringAsIntValue oneOneTwo
set_layerpair_constraint -group foundry -layer1 Metal3 -layer2 Via3 \
-constraint minViaExtension -hardness hard -ValueArrayValue { 0.04 0.06 0.08}
```

The following example sets the minimum via extension for `Metal4` over `Via3`. Two opposite sides must have a minimum extension of 0.04, and two remaining non-diagonal sides must have a minimum extension of 0.05 and 0.06. Any diagonal edges must have a minimum extension of 0.07.

```
set_constraint_parameter -name viaExtensionType -StringAsIntValue diagonal
set_layerpair_constraint -group foundry -layer1 Metal4 -layer2 Via3 \
-constraint minViaExtension -hardness hard -ValueArrayValue { 0.04 0.05 0.06 0.07}
```

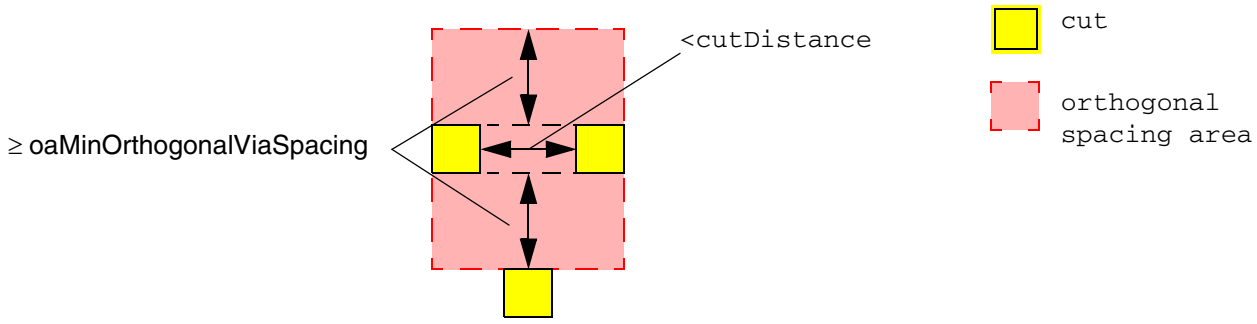
Related Topics

Via Construction Constraints

oaMinOrthogonalViaSpacing

Sets the minimum spacing between an overlapping via cut pair and any other via cut shape based on the distance between the cut pair.

When two cuts overlap, either horizontally or vertically, the distance between them determines the orthogonal spacing, as defined in the lookup table. The spacing applies from opposite edges of their parallel overlap, vertically for a horizontal overlap or horizontally for a vertical overlap. No other cut is allowed in the resultant orthogonal overlap region.



oaMinOrthogonalViaSpacing Quick Reference

Constraint Type	Layer
Value Types	OneDTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`oaMinOrthogonalViaSpacing` constraints have a OneDTblValue in which the lookup key is the spacing between the cut pair.

Format	Example
Tcl	`set_layer_constraint -constraint oaMinOrthogonalViaSpacing \ -row_name width -layer` *`s_cutLayer`* `-OneDTblValue {{`*`f_cutDistance f_space`*`}…}`
LEF	`SPACINGTABLE ORTHOGONAL WITHIN` *`f_cutDistance`* `SPACING` *`f_space`*

Format	Example

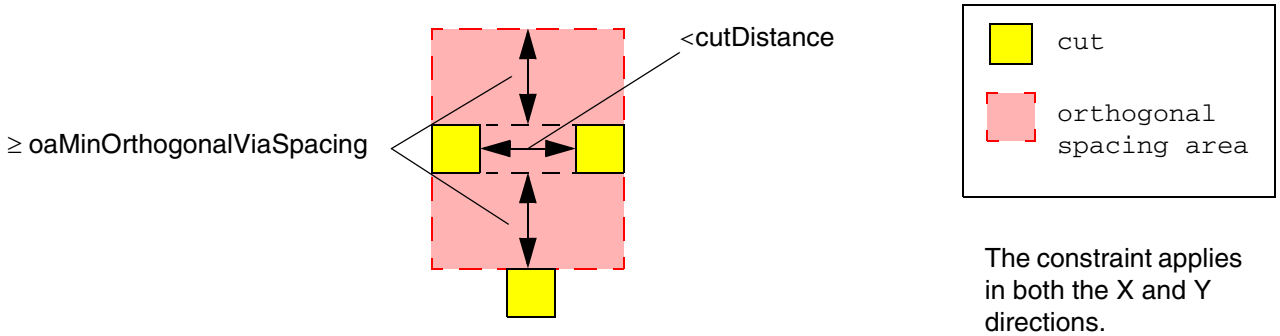
Virtuoso	`spacingTables(`
	`( minOrthogonalViaSpacing` *`s_cutLayer`*
	`  ( ( "distance" nil nil )  )`
	`  (` *`f_cutDistance f_space`* `)`
	`…`
	`)`

Examples

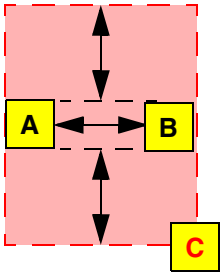
This example sets the required orthogonal spacing between overlapping via cut pairs and other vias as follows:

- If the via cut pair overlap is less than or equal to 0.4, then the orthogonal spacing to other vias must be greater than or equal to 0.75.
- If the via cut pair overlap is greater than 0.4 but less than or equal to 0.6, then the orthogonal spacing to other vias must be greater than or equal to 0.90.

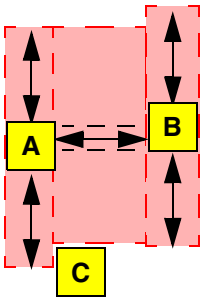
oaMinOrthogonalViaSpacing Examples



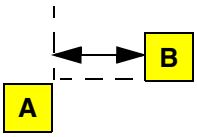
The constraint applies in both the X and Y directions.



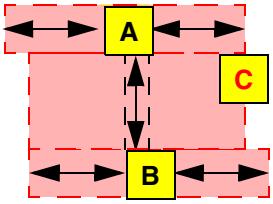
a) Violation; cut C overlaps the orthogonal spacing area required by cuts A and B.



a) OK; cut C is outside the orthogonal spacing area required for cuts A and B.



c) OK; constraint does not apply because there is zero overlap between cuts A and B.



d) Violation; cut C overlaps the orthogonal spacing area required for cuts A and B.

Related Topics

Via Construction Constraints

oaMinParallelViaClearance

Specifies the clearance between cut shapes on one layer with respect to cut shapes on an adjacent layer that have parallel edges with an overlap greater than 0. This constraint applies only when the cuts do not share the same metal shaped on the layer between them.

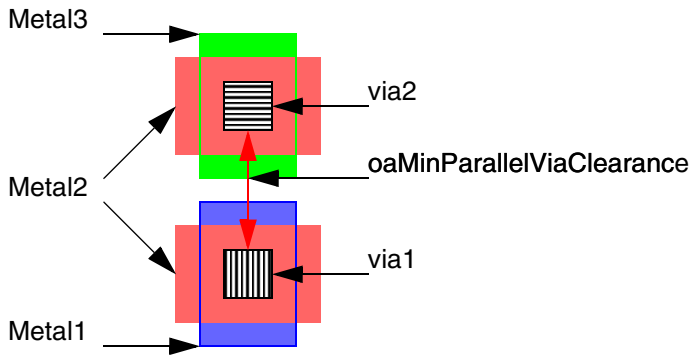
oaMinParallelViaClearance Quick Reference

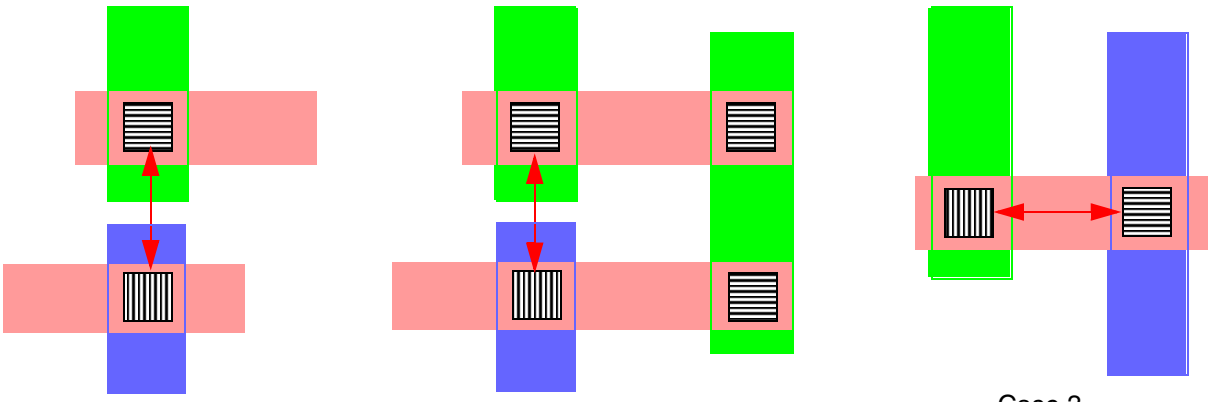
Constraint Type	Layer array
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`oaMinParallelViaClearance` constraints have a `Value` that represents the minimum required clearance, in user units, between the cut shapes.

oaMinParallelViaClearance Example





Case 1
Different net
Constraint always applies

Case 2
Same net, different islands
No common metal
Constraint applies

Case 3
Same net, same island
One layer of common metal
Constraint does not apply

Related Topics

Via Construction Constraints

oaMinViaClearance

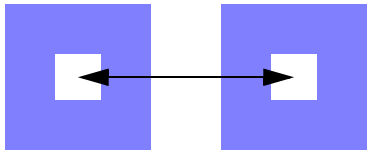
Sets the minimum spacing, either edge-to-edge or center-to-center, between via cuts on the specified layers, either absolutely or conditionally, based upon the net the cuts are on, how a via stack is constructed, or the area of the cuts.

oaMinViaClearance Quick Reference

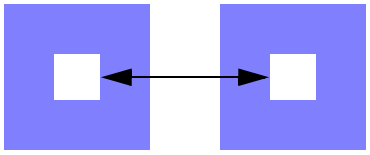
Constraint Type	Layer pair
Value Types	`Value`, `OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Parameters

- `oaCenterToCenter` (`BoolValue`, optional) determines how cut clearance is measured. If set to `true`, cut clearance is measured from the center of one cut to the center of the other cut. If not specified or set to `false`, cut clearance is measured edge-to-edge.



`oaCenterToCenter = true`



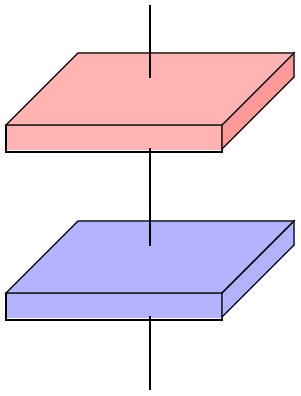
`oaCenterToCenter = false`
`(default, edge-to-edge)`

- `oaConnectivityType` (`StringAsIntValue`, optional) specifies the shapes to which this constraint applies, based on their connectivity, as given in Table on page 921.

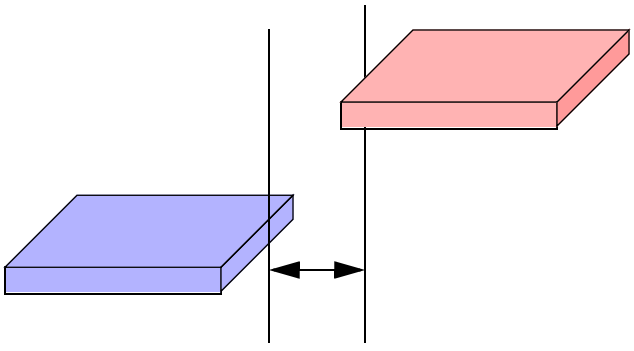
oaConnectivityType Parameter Values

String	Integer Value	Constraint applies to:
`anyConnectivity`	0	(Default) Any connectivity
`sameNetConnectivity`	1	Same net only
`sameIslandConnectivity`	2	Via cuts connected by contiguous same metal
`directConnectShapesConnectivity`	3	Via cuts directly connected by the same metal
`sameViaConnectivity`	4	Via cuts overlapped by a single metal shape from above and by another single metal shape from below

- `oaStack` (`BoolValue`, optional) controls whether via cuts on two different layers can be stacked (`true`) if the cuts are aligned. If `false` or not specified, via cuts cannot be stacked.



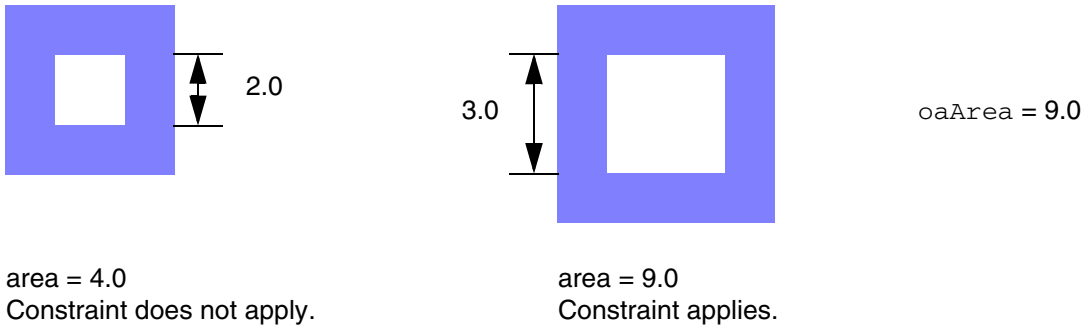
`oaStack` = `true`, cuts may overlap if their centers are aligned.



`oaStack` = `false`, cuts cannot overlap and the minimum clearance applies.

- `oaArea` (`Value`) specifies that the constraint applies only if the area of a cut is equal to or greater than this value. The value of this parameter is assumed to be greater than the

default clearance value. *This parameter is currently not supported by Space-based Router and Chip Optimizer*.



- `exceptConnectivityType` (`IntValue`, optional) specifies that the constraint does not apply to via cuts with certain connectivity types, as given in the following table.

Table 17-1 exceptConnectivityType Parameter Values

Integer Value	Equivalent to:	Constraint does not apply to:
0	`exceptNoConnectivityType`	(Default) Via cuts with no connectivity
1	`exceptSameNetConnectivity`	Via cuts on the same net
2	`exceptSameMetalConnectivity`	Via cuts on a contiguous same metal

Note: `oaConnectivityType` and `exceptConnectivityType` are mutually exclusive.

- `enclosingLayer` (`LayerValue`, optional) specifies the enclosing layer for `viaEdgeTypeParam` extension checks.
- `viaEdgeTypeParam` (`IntValue`, optional) specifies that the constraint applies only to the via cut edges of this type on the first cut layer (the type is defined in the `viaEdgeType` constraint).
- `sizeBy` (`IntValue`, optional) specifies that the constraint is applied after sizing all the edges of the via cut on the first cut layer by this value. If both `enclosingLayer` and `sizeBy` are specified, the spacing is measured between the geometric AND (or intersection) of the enclosing layer extension and the via extended by `sizeBy`. In other

words, the spacing is measured from the enclosing metal edge or the expanded via cut edge, whichever is at a shorter distance from the via cut.

- `extendBy` (`IntValue`, optional) specifies that an extension equal to this value is applied to the via cut edges that satisfy `viaEdgeTypeParam` before the constraint is applied. If `sizeBy` is also specified, the extension is applied after sizing the cuts.
- `cutClass` (`DualValue`, optional) specifies the cut class to which the constraint applies on the first cut layer.
- `otherCutClass` (`DualValue`, optional) specifies the cut class to which the constraint applies on the second cut layer.
- `overlapNotAllowed` (`BoolValue`, optional) specifies whether to show violations for overlapped or stacked vias when the minimum spacing between via cuts on the specified layers is `0`. If set to `true`, a violation is reported for overlapped or stacked vias. By default, the parameter value is `false`.

  However, there is an exception to this parameter. If the `oaStack` parameter is specified and the centers of the overlapped vias are aligned, then no violation is reported.

Value Types

`oaMinViaClearance` constraints have the following value types:

- `Value`

  Specifies the minimum spacing, in user units.

Format	Example

Tcl	`[set_constraint_parameter -name oaConnectivityType \`
	`-StringAsIntValue {anyConnectivity| sameNetConnectivity | \`
	`sameIslandsConnectivity | sameViaConnectivity}]`
	`[set_constraint_parameter -name oaStack -BoolValue {true|false}]`
	`[set_constraint_parameter -name oaCenterToCenter \`
	`-BoolValue {true|false}]`
	`set_layerpair_constraint -constraint oaMinViaClearance \`
	`-layer1 *f_cutLayer1* -layer2 *s_cutLayer2* -Value *f_spacing*`

- `OneDTblValue`

  Specifies the minimum spacing that changes according to the width of the wider of the two shapes. The width of the wider shape is the lookup key for the table. The table value is the minimum spacing. The width of a shape is defined as the smaller of the shape's two dimensions.

Examples

- Example 1: oaMinViaClearance with oaStack and oaConnectivityType
- Example 2: oaMinViaClearance with enclosingLayer, viaEdgeTypeParam
- Example 3: oaMinViaClearance with enclosingLayer, viaEdgeTypeParam, sizeBy, extendBy, cutClass, and otherCutClass

Example 1: oaMinViaClearance with oaStack and oaConnectivityType

```
set_layerpair_constraint -constraint oaMinViaClearance \
  -layer1 via1 -layer2 via2 -Value 0.5
set_constraint_parameter -name oaConnectivityType \
  -StringAsIntValue sameNetConnectivity
set_constraint_parameter -name oaStack -BoolValue true
set_layerpair_constraint -constraint oaMinViaClearance \
  -layer1 cont1 -layer2 cont2 -Value 0.3
```

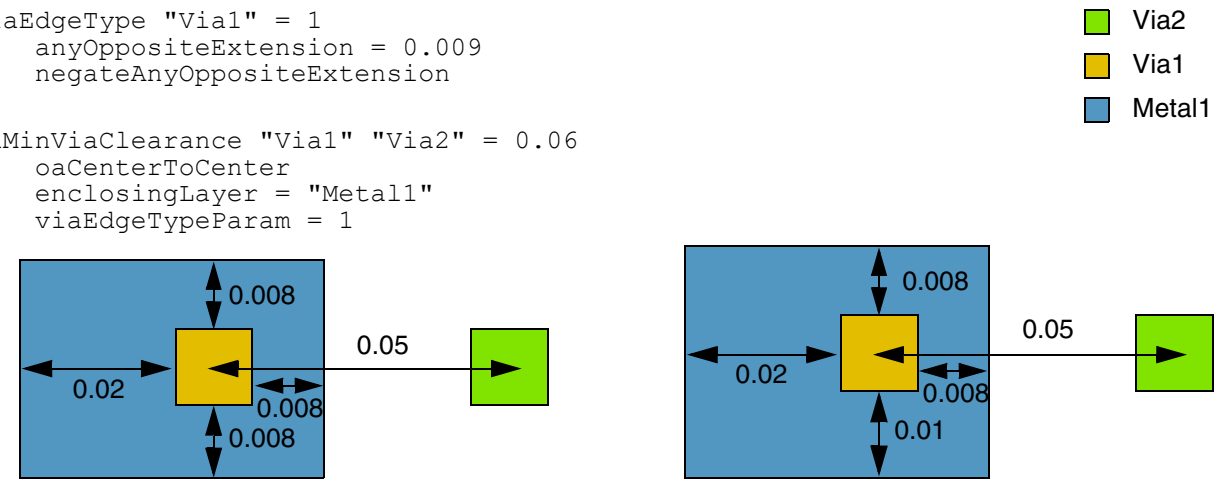
Sets the minimum via cut clearance on `via1` and `via2` to `0.5`. It also sets the minimum via cut clearance on `cont1` and `cont2` to `0.3` when the cuts are on the same net and stacked vias are aligned.

Example 2: oaMinViaClearance with enclosingLayer, viaEdgeTypeParam

```
set_constraint_parameter -name anyOppositeExtension -Value 0.009
set_constraint_parameter -name negateAnyOppositeExtension -BoolValue true
set_layer_constraint -layer Via1 -constraint viaEdgeType  -IntValue 1

set_constraint_parameter -name viaEdgeTypeParam -IntValue 1
set_constraint_parameter -name enclosingLayer -LayerValue Metal1
set_constraint_parameter -name oaCenterToCenter -BoolValue true
set_layerpair_constraint -layer1 Via1 -layer2 Via2 \
    -constraint oaMinViaClearance -Value 0.06

check_layerpair_space -lpp1 { Via1 } -lpp2 { Via2 } -diff_net true -same_net true
```

Sets the center-to-center spacing between a `Via1` via cut with `viaEdgeType` 1 (the constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009), enclosed by `Metal1`, and a `Via2` cut to at least 0.06.

```
viaEdgeType "Via1" = 1
    anyOppositeExtension = 0.009
    negateAnyOppositeExtension

oaMinViaClearance "Via1" "Via2" = 0.06
    oaCenterToCenter
    enclosingLayer = "Metal1"
    viaEdgeTypeParam = 1
```



a) The constraint does not apply because there exist two opposite edges with extension 0.008 (that is, with extension less than or equal to 0.009).

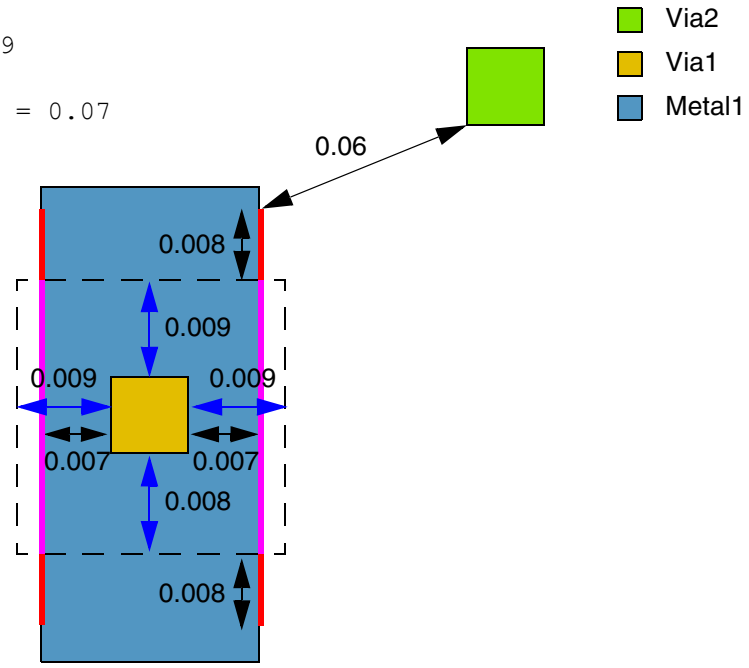
b) FAIL. The constraint applies because the edges with extension 0.008 (that is, with extension less than or equal to 0.009) are not opposite edges. However, the center-to-center spacing between the two via cuts is 0.05 (<0.06).

Example 3: oaMinViaClearance with enclosingLayer, viaEdgeTypeParam, sizeBy, extendBy, cutClass, and otherCutClass

```
set_constraint_parameter -name anyOppositeExtension -Value 0.009
set_constraint_parameter -name edgeExtension -Value 0.009
set_layer_constraint -layer Via1 -constraint viaEdgeType -IntValue 2

set_constraint_parameter -name cutClass -DualValue {0.032 0.032}
set_constraint_parameter -name viaEdgeTypeParam -IntValue 2
set_constraint_parameter -name enclosingLayer -LayerValue Metal1
set_constraint_parameter -name extendBy -Value 0.008
set_constraint_parameter -name sizeBy -Value 0.009
set_layerpair_constraint -layer1 Via1 -layer2 Via2 \
    -constraint oaMinViaClearance -Value 0.07

check_layerpair_space -lpp1 { Via1 } -lpp2 { Via2 } -diff_net true -same_net true
```

Specifies that the edge-to-edge spacing between a `Via1` via cut edge with `viaEdgeType` 2 and a `Via2` via cut must be at least 0.07 when the `Via1` via cut, enclosed by `Metal1`, is sized by 0.009 and extended by 0.008. The constraint applies only to `Via1` via cuts with opposite edge extensions less than or equal to 0.009 (any edges on those via cuts with extension less than or equal to 0.009 are of `viaEdgeType` 2).

```
viaEdgeType "Via1" = 2
    anyOppositeExtension = 0.009
    edgeExtension = 0.009
oaMinViaClearance "Via1" "Via2" = 0.07
    enclosingLayer = "Metal1"
    viaEdgeTypeParam = 2
    extendBy = 0.008
    sizeBy = 0.009
```



FAIL. The constraint applies because the Via1 cut has a pair of opposite edges (left and right) with extension 0.007 (that is, with extension less than or equal to 0.009). The pink edges indicate the intersection of Via1 cut and Metal1 shape after the via cut is sized by 0.009. The pink edges are extended by 0.008 (the red edges) before the distance between the two via cuts is measured (that is, spacing is measured between "the AND of the Metal1 shape and the Via1 cut sized by 0.009, which is then extended by 0.008" and the nearest edge of the Via2 cut), and this distance is found to be 0.06 (<0.07). Therefore, a violation occurs.

Related Topics

Via Construction Constraints

oaMinViaSpacing

Sets the minimum spacing, either edge-to-edge or center-to-center, between via cuts on the specified layer, either absolutely or conditionally, based on the net the cuts are on.

oaMinViaSpacing Quick Reference

Constraint Type	Layer
Value Types	Value, OneDTblValue
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Types

Value	Specifies the minimum spacing, in user units.
OneDTblValue	Specifies the minimum spacing that changes according to the width of the wider of the two shapes. The width of the wider shape is the lookup key for the table. The table value is the minimum spacing. The width of a shape is defined as the smaller of the shape's two dimensions.

Required Parameters

None

Optional Parameters

distanceMeasureType	Specifies the spacing measurement method as Euclidean (0, default) or Manhattan (1). For information about this parameter, see Euclidean and Manhattan Spacing Constraints. Type: IntValue

`oaCenterToCenter`

Determines how cut clearance is measured. If set to `true`, cut clearance is measured from the center of one cut to the center of the other cut. If not specified or set to `false`, cut clearance is measured edge-to-edge.

Type: `BoolValue`

`oaConnectivityType`

Specifies the shapes to which this constraint applies, based on their connectivity. This parameter is mutually exclusive with `exceptConnectivityType`.

String	Integer Value
`anyConnectivity` (default) Constraint applies to any connectivity	0
`sameNetConnectivity` Constraint applies to same net only	1
`sameIslandConnectivity` Constraint applies to via cuts connected by contiguous same metal	2
`directConnectShapesConnectivity` Constraint applies to via cuts directly connected by the same metal	3
`sameViaConnectivity` Constraint applies to via cuts with the same metal above and same metal below	4

Type: `StringAsIntValue`

`exceptConnectivityType`

Specifies the vias for which this constraint does not apply, based on their connectivity. This parameter is mutually exclusive with `oaConnectivityType`.

String	Integer Value
`sameNetConnectivity`	1
Constraint does not apply to same net only	
`sameIslandConnectivity`	2
Constraint does not apply to via cuts connected by contiguous same metal	
`directConnectShapesConnectivity`	3
Constraint does not apply to via cuts directly connected by the same metal	
`sameViaConnectivity`	4
Constraint does not apply to via cuts with the same metal above and same metal below	

Type: `StringAsIntValue`

`oaArea`

Specifies that the constraint applies only if the area of a cut is equal to or greater than this value. The value of this parameter is assumed to be greater than the default clearance value. *This parameter is not currently supported by Space-based Router and Chip Optimizer*.

Type: `Value`

`parallelRunLength`

Specifies that the constraint applies only if the parallel run length (prl) between two via cuts is greater than or equal to this value. Both positive and negative values are allowed. This parameter is mutually exclusive with `exactParallelRunLength`.

If multiple `oaMinViaSpacing` constraints are defined in an AND constraint group, the constraint is applied by evaluating the prl values, specified in the ascending order, to locate the first prl value for which the actual prl is greater than or equal to the parameter value. As a result, only one constraint in an AND constraint group applies. For example, in an AND constraint group with prl values specified as -1.0, -0.05, 0, 0.05, 1, the constraint value that applies is determined as follows:

Actual length (prl)	**Use constraint value for length**
<-1	The constraint does not apply.
>=-1 and <-0.05	-1.0
>=-0.05 and <0	-0.05
>=0 and <0.05	0
>=0.05 and <1.0	0.05
>=1.0	1.0

Type: `Value`

`exactParallelRunLength`

Specifies that the constraint applies only if the parallel run length is this value, in user units. This parameter is mutually exclusive with `parallelRunLength`.

Type: `Value`

`exactSpacing`

Specifies whether the constraint value is an exact spacing (`true`) or a minimum spacing (`false`/default). Must be used with the `exactParallelRunLength` parameter.

Type: `BoolValue`

`sameMask`
Specifies whether the constraint applies between cut shapes on the same mask (`true`) or between all shapes on the given cut layer (`false`/default).

Type: `BoolValue`

`diffMask`
Specifies whether the constraint applies between cut shapes on different masks (`true`) or between all shapes on the given cut layer (`false`/default).

Type: `BoolValue`

`bothCuts`
Specifies that the constraint applies only when both cuts are of the type specified by `viaEdgeTypeParam` (`true`). The default is `false`.

Type: `BoolValue`

`viaEdgeTypeParam`
Specifies that the constraint applies only to via edges of the specified type.

Different via edge types are specified using separate viaEdgeType constraints.

Type: `IntValue`

`enclosingLayer`
Specifies the enclosing layer for via edge type extension checks.

Type: `LayerValue`

`sizeBy`
Specifies that cuts are to be increased in size by this value before the spacing check is performed. If both `enclosingLayer` and `sizeBy` are specified, the spacing is checked between the AND of the `enclosingLayer` extension and the via extended by the `sizeBy` value.

Type: `Value`

`oaSpacingDirection`
Specifies that the constraint applies only to extensions measured in this direction.

Valid values:

`anyDirection`	`0`	Horizontally and vertically (default)
`horizontalDirection`	`1`	Horizontally only
`verticalDirection`	`2`	Vertically only

Type: `IntValue`

`exceptExactAligned`	Specifies that the constraint does not apply to cuts that are aligned (`true`). The default is `false`. Type: `BoolValue`

Examples

- Example 1: oaMinViaSpacing with oaConnectivityType
- Example 2: oaMinViaSpacing with exactParallelRunLength, enclosingLayer, viaEdgeTypeParam, bothCuts, and sizeBy

Example 1: oaMinViaSpacing with oaConnectivityType

```
set_constraint_parameter -name oaConnectivityType -IntValue 1
set_layer_constraint -constraint oaMinViaSpacing \
  -layer via1 -Value 0.3
```

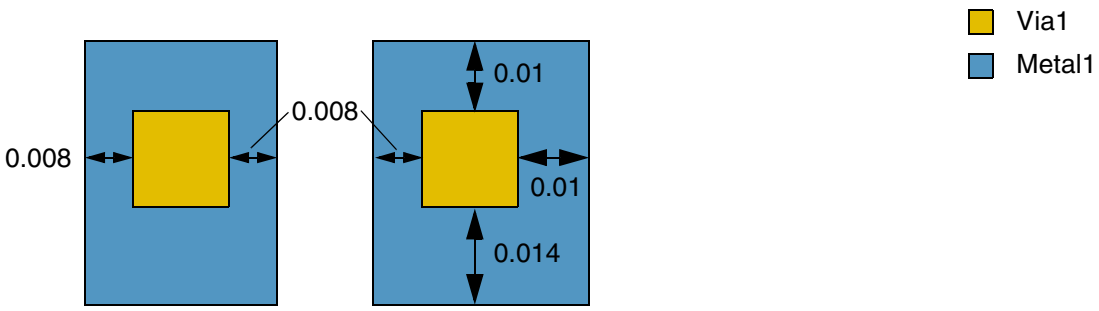
Sets the minimum via spacing on `via1` to 0.3 when the cuts are on the same net.

Example 2: oaMinViaSpacing with exactParallelRunLength, enclosingLayer, viaEdgeTypeParam, bothCuts, and sizeBy

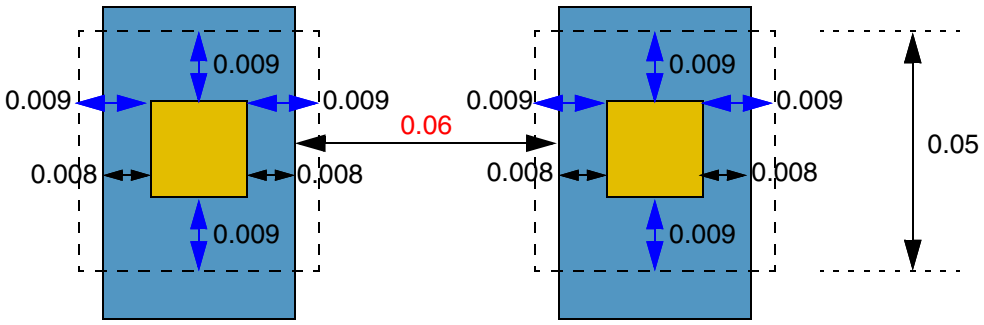
```
set_constraint_parameter -name anyOppositeExtension -Value 0.009
set_layer_constraint -layer Via1 -constraint viaEdgeType -IntValue 2

set_constraint_parameter -name exactParallelRunLength -Value 0.05
set_constraint_parameter -name enclosingLayer -LayerValue Metal1
set_constraint_parameter -name viaEdgeTypeParam -IntValue 2
set_constraint_parameter -name bothCuts -BoolValue true
set_constraint_parameter -name sizeBy -Value 0.009
set_layer_constraint -constraint oaMinViaSpacing -layer Via1 -Value 0.07
```

Specifies that the spacing between two via cuts expanded by 0.009 must be at least 0.07 if the following conditions are met:

- Both via cuts have Metal1 extensions less than or equal to 0.009 on a pair of opposite edges.
- The parallel run length between the via cuts is 0.05.



a) The constraint does not apply because only the left via cut has a pair of opposite edges with 0.008 Metal1 extension (<0.009). Because `bothCuts` is specified, `viaEdgeTypeParam` must also be satisfied for the right via cut.



b) FAIL. Both via cuts have a pair of opposite edges with 0.008 Metal1 extension (<0.009), and the parallel run length between the via cuts expanded by 0.009 is 0.05. However, the spacing between the enclosing metal edges (the AND of the Metal1 enclosing layer extension and the via cut expanded by 0.009) is 0.06 (<0.07).

Related Topics

Via Construction Constraints

rectangularLargeViaArraysAllowed

Specifies whether rectangular via arrays are permitted within a larger via array or only the default square via arrays are allowed.

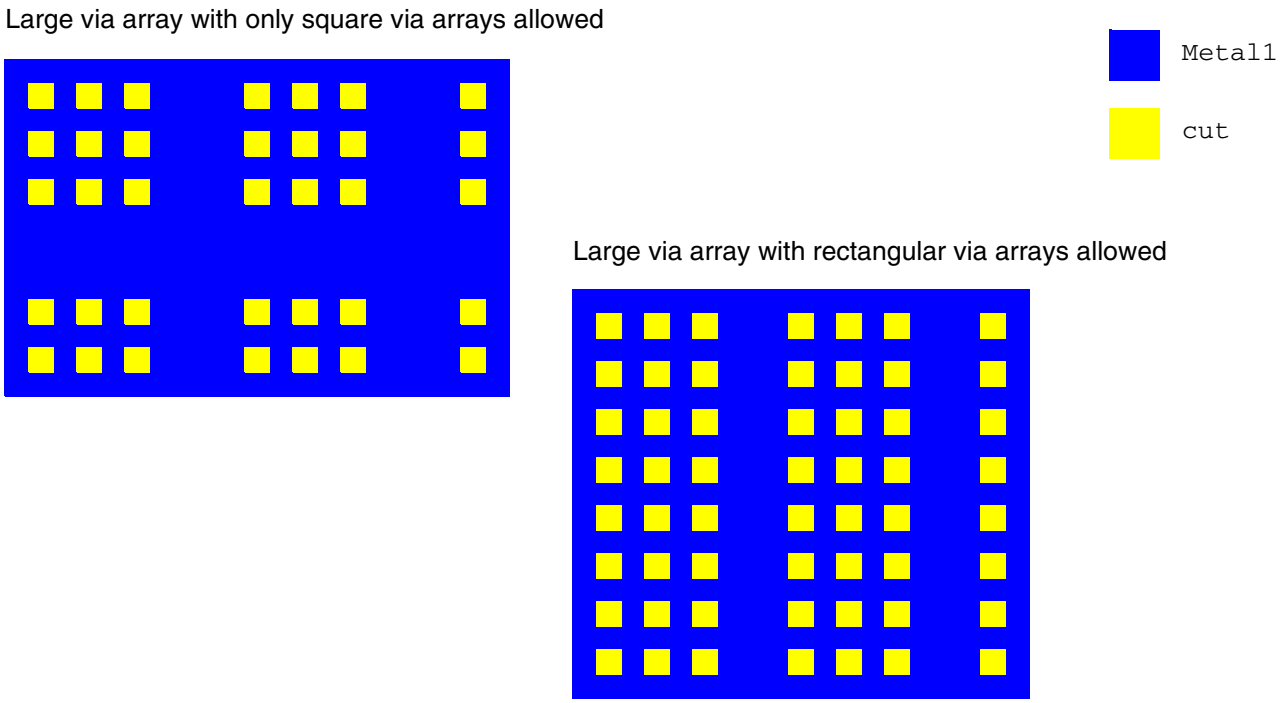
rectangularLargeViaArraysAllowed Quick Reference

Constraint Type	Layer
Value Types	BoolValue
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

`rectangularLargeViaArraysAllowed` constraints have a BoolValue. By default, only square via arrays are allowed. When this constraint permits rectangular via arrays, both square and rectangular via arrays are enabled. As shown in the following figure, the via can be cut off at the end of a wire; this results in incomplete square via arrays, but is not a violation of this constraint.

rectangularLargeViaArraysAllowed Examples



Parameters

- `cutClass` (`DualValue`, custom) The constraint only applies to cut shapes whose width and height are equal to the values in the `DualValue` pair, specified in that order. `parallelOverlap` (`BoolValue`, optional) The constraint applies only when there is a parallel edge overlap > 0.

Examples

This example allows rectangular via arrays on the layer `cut1`; prohibits rectangular via arrays on the layer `cut2`.

Format	Example

Tcl	`set_layer_constraint -constraint rectangularLargeViaArraysAllowed \`
	`  -layer cut1 -BoolValue true`
	`set_layer_constraint -constraint rectangularLargeViaArraysAllowed \`
	`  -layer cut2 -BoolValue false`

Format	Example

LEF	`LAYER cut1`
	`  ARRAYSPACING LONGARRAY …`
	`LAYER cut2`
	`  ARRAYSPACING …`

Virtuoso	`spacings(`
	`  (largeRectViaArrayAllowed "cut1" t)`
	`  (largeRectViaArrayAllowed "cut2" nil)`
	`  )`

Related Topics

Via Construction Constraints

sameNetLargeViaSpacing

This constraint is obsolete. Use minCutClassSpacing with the `oaConnectivityType` parameter set for `sameNetConnectivity`.

viaBarAdjacentSpacing

This constraint is obsolete. Use minAdjacentViaSpacing instead with the `cutClass` parameter to specify the via dimensions.

viaEdgeType

Defines the types of via edges that are referenced by other constraints as parameters. The constraint applies to the cut layer.

viaEdgeType Quick Reference

Constraint Type	Layer
Value Types	`IntValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

The `viaEdgeType` constraint has an `IntValue` that identifies the types of edges specified by this constraint.

Parameters

- `edgeExtension` (`Value`, optional) specifies that via edges with extensions less than or equal to this value for the entire edge, will be assigned to this `viaEdgeType`.
- `anyOppositeExtension` (`Value`, optional) specifies that the constraint applies only to cuts with two opposite edges with extensions less than or equal to this value for the entire edge.
- `negateEdgeExtension` (`BoolValue`, optional) applies only when `edgeExtension` is set. If this parameter is `true`, all edges that do not satisfy the `edgeExtension` parameter, are assigned to this `viaEdgeType`.
- `negateAnyOppositeExtension` (`BoolValue`, optional) applies only when `anyOppositeExtension` is set. If this parameter is `true`, the constraint applies only to cuts that do not satisfy the `anyOppositeExtension` parameter.

Examples

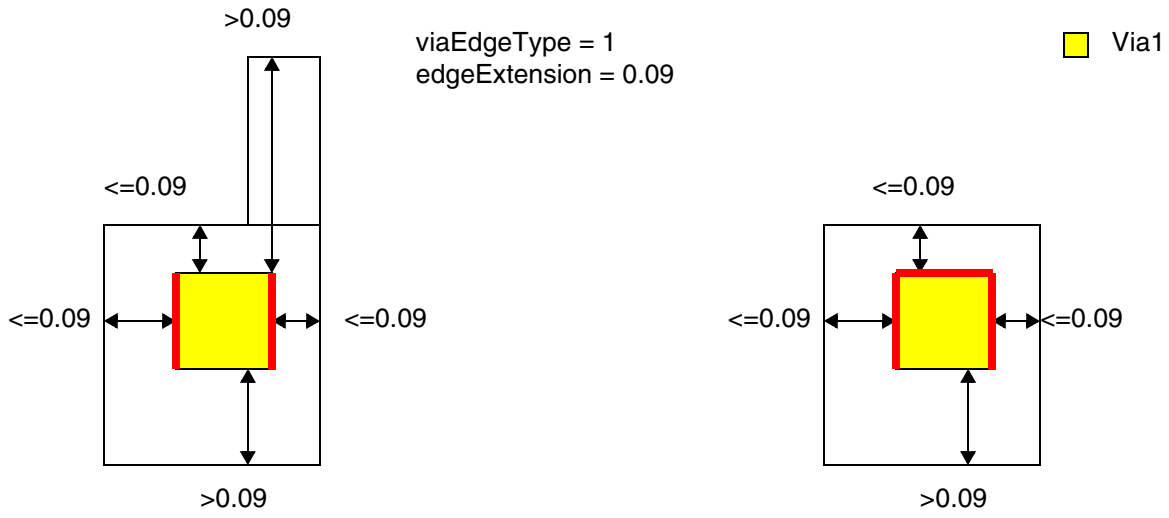
- Example 1 — viaEdgeType with edgeExtension only
- Example 2 — viaEdgeType with edgeExtension and anyOppositeExtension

Example 1 — viaEdgeType with edgeExtension only

In this example, via edges with extensions less than or equal to 0.09 user units, will be assigned to `viaEdgeType` 1.

Format	Example
Tcl	`set_constraint_parameter -name edgeExtension -Value 0.09` `set_layer_constraint -layer Via1 -constraint viaEdgeType -IntValue 1`

Illustration for viaEdgeType with edgeExtension only



Edges shown in red are `viaEdgeType` 1.
The top via edge for the via on the left is not considered to be `viaEdgeType` 1 because part of that edge has extension > 0.09.

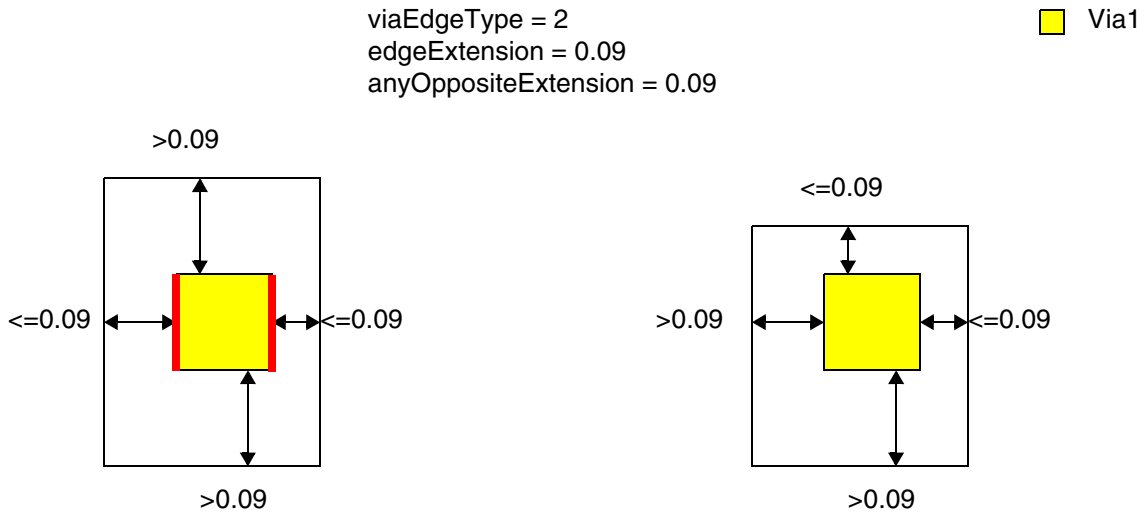
Example 2 — viaEdgeType with edgeExtension and anyOppositeExtension

In this example, only cuts with two opposite extensions that are less than or equal to 0.09 user units are considered. For those cuts, via edges with extensions less than or equal to 0.09 user units will be assigned to `viaEdgeType` 2.

Format	Example

Tcl	`set_constraint_parameter -name edgeExtension -Value 0.09`
	`set_constraint_parameter -name anyOppositeExtension -Value 0.09`
	`set_layer_constraint -layer Via1 -constraint viaEdgeType -IntValue 2`

Illustration for viaEdgeType with edgeExtension and anyOppositeExtension



a) This cut has two opposite edges with extensions <= 0.09, so the constraint applies and the edges in red are `viaEdgeType` 2.

b) This cut does not have two opposite edges with extensions <= 0.09, so the constraint does not apply.

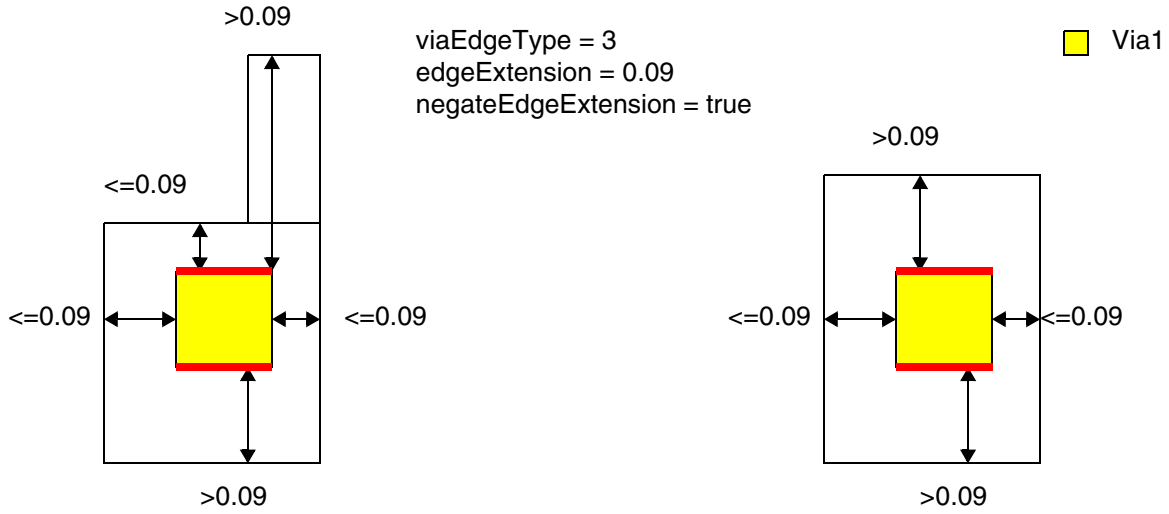
Example 3 — viaEdgeType with edgeExtension and negateEdgeExtension

In this example, all via edges that do not have extensions less than or equal to 0.09 user units, are assigned to `viaEdgeType` 3.

Format	Example

Tcl	`set_constraint_parameter -name edgeExtension -Value 0.09`
	`set_constraint_parameter -name negateEdgeExtension -BoolValue true`
	`set_layer_constraint -layer Via1 -constraint viaEdgeType -IntValue 3`

Illustration for viaEdgeType with edgeExtension and negateEdgeExtension



Edges shown in red are `viaEdgeType` 3.

Example 4 — viaEdgeType with edgeExtension, anyOppositeExtension and negateAnyOppositeExtension

In this example, the constraint applies only to cuts that do not have two opposite edges with extension less than or equal to 0.09 user units. For those cuts, via edges with extensions less than or equal to 0.09 user units, will be assigned to `viaEdgeType` 4.

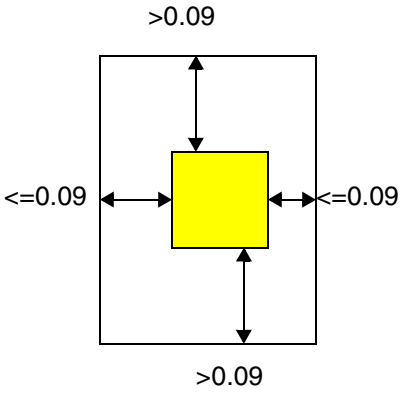
Format	Example

Tcl	`set_constraint_parameter -name edgeExtension -Value 0.09`
	`set_constraint_parameter -name anyOppositeExtension -Value 0.09`
	`set_constraint_parameter -name negateAnyOppositeExtension \`
	`  -BoolValue true`
	`set_layer_constraint -layer Via1 -constraint viaEdgeType -IntValue 4`

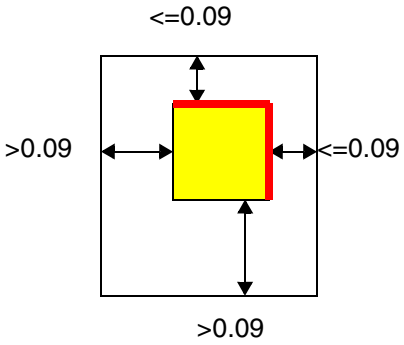
Illustration for viaEdgeType with edgeExtension, anyOppositeExtension and negateAnyOppositeExtension

viaEdgeType = 4
edgeExtension = 0.09
anyOppositeExtension = 0.09
negateAnyOppositeExtension =true

Via1



a) This cut has two opposite edges with extensions <= 0.09, so the constraint does not apply.



b) This cut does not have two opposite edges with extensions <= 0.09, so the constraint applies. Edges in red are `viaEdgeType` 4.

Related Topics

Via Construction Constraints

viaEdgeType

viaStackingAllowed

Specifies whether you can stack the two cut layers. This constraint should be specified for cut layers only. A via is considered to be in a stack with another via if the cut of one via overlaps any part of the cut of the other via.

This constraint is symmetric, implying that when via stacking is allowed for via1 and via2, it is also allowed for via2 and via1.

viaStackingAllowed Quick Reference

Constraint Type	Layer pair
Value Types	`BoolValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Type

The `viaStackingAllowed` constraint has a `BoolValue`. When set to `true`, via stacking is allowed for the two cut layers. When set to `false`, via stacking is not allowed for the two layers.

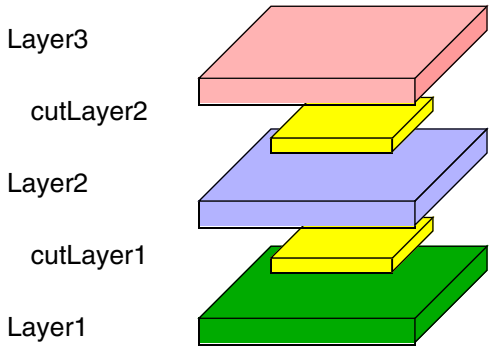
Examples

This example permits stacking of `cutLayer1` and `cutLayer2`, as shown in the following figure.

Format	Example
Tcl	`set_layerpair_constraint -layer1 cutLayer1 -layer2 cutLayer2 \` `  -constraint viaStackingAllowed -BoolValue true`

Virtuoso	`spacings (`
	`(stackable "cutLayer1" "cutLayer2" t )`
	`)`

viaStackingAllowed Example



Setting viaStackingAllowed for cutLayer1 and cutLayer2 to false would prevent the stacking shown here.

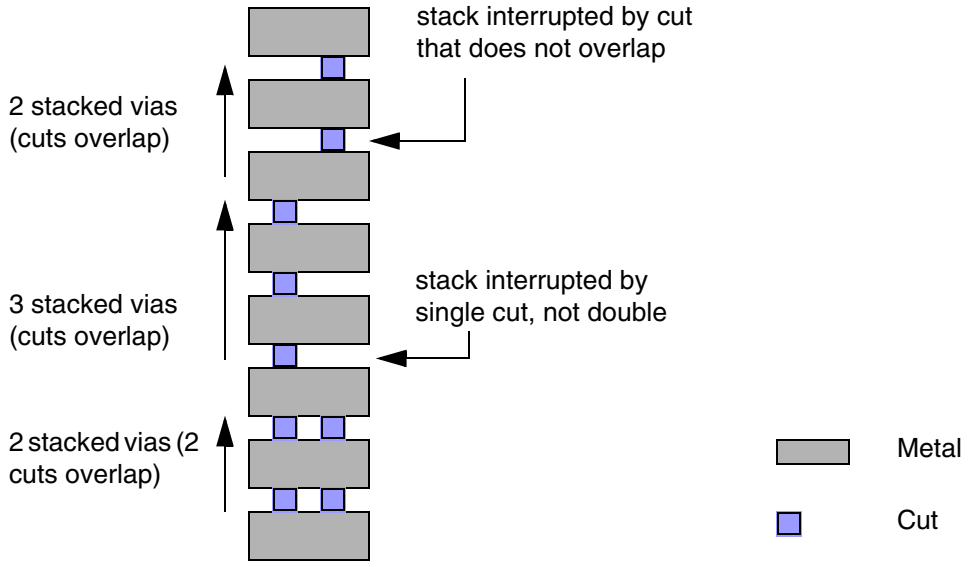
Related Topics

Via Construction Constraints

viaStackLimit

Specifies the maximum number of single-cut stacked vias that are allowed in one continuous stack. A via is considered to be in a stack with another via if the cut of one via overlaps any part of the cut of the other via. A double-cut or larger via interrupts the stack.

An optional parameter specifies whether stacked vias can be single-cut or must be multi-cut.



Two optional parameters may be used to specify a range of layers for which this constraint applies. If the range is not given, the constraint applies for all layers.

viaStackLimit Quick Reference

Constraint Type	Simple
Value Types	IntValue
Database Types	Design, Technology
Scope	design, foundry
Category	Via Construction

Value Types

The `viaStackLimit` constraint has an `IntValue` that specifies the maximum number of single-cut stacked vias that are allowed on top of each other in one continuous stack.

Format	Example

Tcl	`[set_constraint_parameter -name lowerLayer -LayerValue s_mlayer1`
	`set_constraint_parameter -name upperLayer -LayerValue s_mlayer2]`
	`[set_constraint_parameter -name oaNoSingleCutVia \`
	`  -BoolValue {true|false}]`
	`set_constraint -constraint viaStackLimit -IntValue i_count`

LEF	`MAXVIASTACK i_count [RANGE s_mlayer1 s_mlayer2] ;`

Virtuoso	`viaStackingLimits (`
	`  (i_count ['noSingleCut] [s_mlayer1 s_mlayert2])`
	`)`

Parameters

The `viaStackLimit` constraint has the following arguments:

- `lowerLayer` (`LayerValue`, optional) and `upperLayer` (`LayerValue`, optional) When `upperLayer` and `lowerLayer` are specified, the constraint applies only to stacking vias in the range between the two layers. If these are not given, the constraint applies to all layers. The layers must be metal layers.
- `oaNoSingleCutVia` (`BoolValue`, optional) chooses whether stacked vias can be single-cut (`false`, default) or must be multi-cut (`true`). If set to `true`, via stacks taller than the constraint value must consist of all multiple-cut vias.

Example

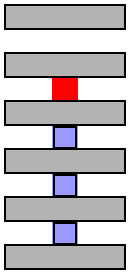
This example sets the maximum stack to be 3. The following figure shows stacked via examples and shows results when this constraint is applied with varied settings for the `oaNoSingleCutVia` parameter.

Format	Example
Tcl	`set_constraint -constraint viaStackLimit -IntValue 3`
LEF	`MAXVIASTACK 3 ;`

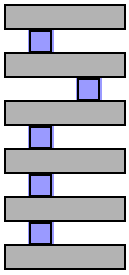
Virtuoso	`viaStackingLimits (`
	`  ( 3 )`
	`)`

viaStackLimit Examples

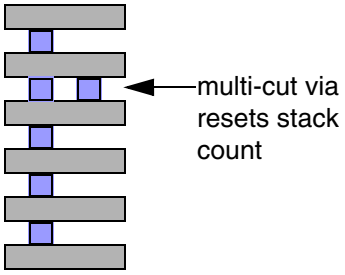
Assume that viaStackLimit = 3.



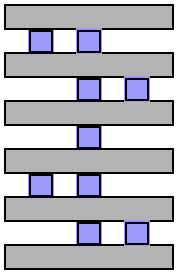
a) Violation
Four stacked vias > viaStackLimit



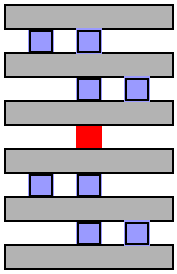
b) OK
The number of consecutive stacked vias <=viaStackLimit



c) OK
The number of consecutive stacked vias <=viaStackLimit



d) OK
By default and when oaNoSingleCutVia = false.



e) Violation
When oaNoSingleCutVia = true, if the number of stacked vias > viaStackLimit, all of the stacked vias must be multi-cuts.

Related Topics

Via Construction Constraints

18

Wire Widening Constraints

Wire widening reduces the probability of opens, but can potentially increase the probability of shorts with neighboring shapes. It improves yield by reducing the likelihood of opens due to defects.

Following are the wire widening constraints:

- wideningMinSplitValue
- wideningTargetWidth

Related Topics

Tcl Constraint Commands

Manage Constraints

wideningMinSplitValue

Specifies the minimum length, in user units, for a split when widening wires.

wideningMinSplitValue Quick Reference

Constraint Type	Layer
Value Types	`Value`
Database Types	Design, Technology
Scope	design, foundry
Category	Wire Widening

Value Type

The `wideningMinSplitValue` constraint has a `Value` that specifies the minimum length, in user units, for a split when widening wires. The minimum default is one track.

Related Topics

Wire Widening Constraints

wideningTargetWidth

Specifies the amount of wire widening to attempt.

wideningTargetWidth Quick Reference

Constraint Type	Layer
Value Types	`OneDTblValue`
Database Types	Design, Technology
Scope	design, foundry
Category	Wire Widening

Value Type

The `wideningTargetWidth` constraint has a `OneDTblValue` that specifies the amount of wire widening to attempt, based on the width of the wire. The default is 110% of the current wire width.

Related Topics

Wire Widening Constraints